ERDC/CERL SR-07-7

Construction Engineering Research Laboratory

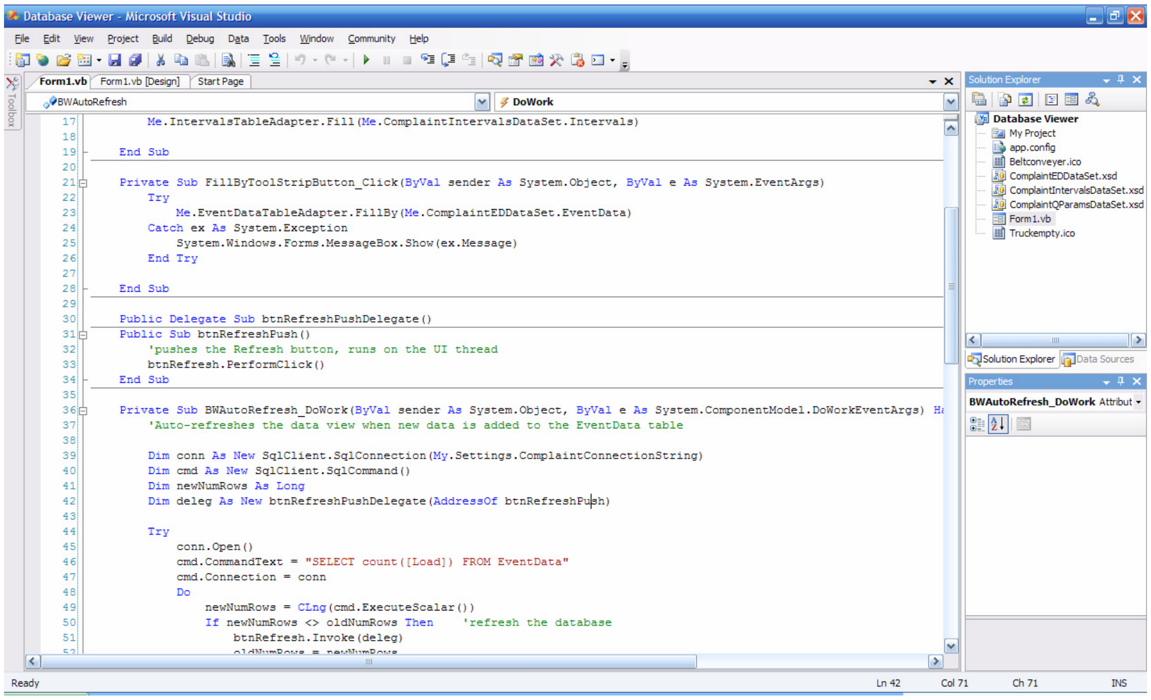
US Army Corps of Engineers®
Engineer Research and Development Center

ERDC-CERL LD-870 Download Program

Programming Manual

Ben Niemoeller and Edward T. Nykaza

May 2007



```
            Me.IntervalsTableAdapter.Fill(Me.ComplaintIntervalsDataSet.Intervals)

        End Sub

        Private Sub FillByToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
            Try
                Me.EventDataTableAdapter.FillBy(Me.ComplaintEDDataSet.EventData)
            Catch ex As System.Exception
                System.Windows.Forms.MessageBox.Show(ex.Message)
            End Try

        End Sub

        Public Delegate Sub btnRefreshPushDelegate()
        Public Sub btnRefreshPush()
            'pushes the Refresh button, runs on the UI thread
            btnRefresh.PerformClick()
        End Sub

        Private Sub BWAutoRefresh_DoWork(ByVal sender As System.Object, ByVal e As System.ComponentModel.DoWorkEventArgs) Ha
            'Auto-refreshes the data view when new data is added to the EventData table

            Dim conn As New SqlClient.SqlConnection(My.Settings.ComplaintConnectionString)
            Dim cmd As New SqlClient.SqlCommand()
            Dim newNumRows As Long
            Dim deleg As New btnRefreshPushDelegate(AddressOf btnRefreshPush)

            Try
                conn.Open()
                cmd.CommandText = "SELECT count([Load]) FROM EventData"
                cmd.Connection = conn
                Do
                    newNumRows = CLng(cmd.ExecuteScalar())
                    If newNumRows <> oldNumRows Then    'refresh the database
                        btnRefresh.Invoke(deleg)
```

ERDC/CERL SR-07-7
May 2007

ERDC-CERL LD-870 Download Program

Programming Manual

Ben Niemoeller and Edward T. Nykaza

Construction Engineering Research Laboratory
U.S. Army Engineer Research and Development Center
PO Box 9005
Champaign, IL 61826-9005

Final report

Prepared for U.S. Army Aberdeen Test Center
Aberdeen Proving Ground, MD 21005-5059

Under Work Unit 0B72D9

Abstract: The U.S. Army Engineer Research and Development Center Construction Engineering Research Laboratory has developed software that interfaces with an array of Larson-Davis Model 870 Environmental Noise Monitors for Aberdeen Test Center. This document explains logic and procedures used while programming the software that are of interest to a programmer looking to modify or expand the functionality of the program. The following topic areas are covered: terminology, time synchronization, and scheduling events. This document will be of interest to those who wish to modify the L-D Download software program. The code, which was written with Microsoft Visual Studio 2005, is included in the appendices.

Contents

Preface

This study was conducted for Aberdeen Test Center (ATC), Aberdeen Proving Ground, MD, under MIPR6FXXR3A563, "R&D of Military Noise Assessment Tools to Support ATC's Noise Program." The ATC technical monitor was Kimberley Fillinger.

The work was performed by the Ecological Processes Branch (CN-N) of the Installations Division (CN), Construction Engineering Research Laboratory (CERL). The CERL Principal Investigator was Edward T. Nykaza. Alan B. Anderson is Chief, CN-N, and Dr. John T. Bandy is Chief, CN. The associated Technical Director was William Severinghaus, CVT. The Deputy Director of CERL is Dr. Kirankumar V. Topudurti and the Director of CERL is Dr. Ilker Adiguzel.

CERL is an element of the U.S. Army Engineer Research and Development Center (ERDC), U.S. Army Corps of Engineers. The Commander and Executive Director of ERDC is COL Richard B. Jenkins, and the Director of ERDC is Dr. James R. Houston.

1 Introduction

Background

Aberdeen Test Center (ATC) in Maryland has one of the top installation environmental noise programs in the nation. The reason they are at the forefront of environmental noise programs is largely due to the trained staff and number of noise monitors they have located in the communities surrounding the installation. Over the past several years, there have been issues with the reliability of the software used to download data from the noise monitors.

In April 2006 ATC funded the U.S. Army Engineer Research and Development Center Construction Engineering Research Laboratory (ERDC/CERL) to replace the outdated software ATC uses to download and manage the data from their noise monitors. The new software developed by ERDC/CERL was built upon software developed for a research project being conducted at Aberdeen Proving Ground and modified to meet the needs of the ATC staff who run the system on a day-to-day basis. The software went into operation in August 2006.

Objectives

The objective of the work was to develop a Windows-based server and database software to download data from ATC's environmental noise monitors.

Approach

This document contains ERDC/CERL's knowledge of the noise monitoring system at Aberdeen Proving Ground. The document explains some of the logic and procedures used to program the software. It covers the following topic areas: terminology, unit list, time synchronization, scheduler, and Larson Davis (LD) unit class variables. The code, which was written with Microsoft Visual Studio 2005, is included in the appendices.

Scope

This report contains the software code and logic for the noise monitoring system and is intended to be used by programmers who wish to update the software. A separate report (ERDC/CERL SR-07-6) contains operational knowledge and is intended to serve as a user's manual.

2 Terminology

This terminology is used to define the structure, hierarchy, and functions of an object-oriented program, in addition to items that are specific to .NET and Windows programming.

Namespace – a level of hierarchy above class. In Visual Basic (VB), all of the code for one project resides in a single namespace. The namespace for the Aberdeen Test Center download program is WindowsApplication1. The namespaces for other programs are L_D_Admin, L_D_Download_1_modem, and Configurator.

Class – A class is a logical grouping of data, along with instructions to perform on that data.

Object – An instance of a class, generated at runtime. When code mentions a specific instance of a class, that instance is often called an object.

Object data type – In .NET, an Object data type is also the most primitive data type; that is, all other data types are inherited from the Object data type.

Field – A field is a variable inside the class that is accessible by all methods of that class. This is called a "class variable" outside the realm of Windows programming. A field can contain a primitive data type or an instance of a class.

Method – The executable code within a class is contained in one or more methods. A method can call other methods, access data in its own class as well as in other classes, and even instantiate new classes. VB has two kinds of methods, a Sub(routine) and a Function. A Sub is a method that does not explicitly return data to the method that called it. A Function must return some data, via a Return statement, to the method that called it.

Local variable – a variable declared inside a method. This is often just called a "variable." The variable, as well as the data contained in it, is lost when the method exits.

Block – the code contained within an If…Then statement or a program loop. The code in a block will be indented.

Block variable – a variable declared inside an If…Then block or a loop. The variable, as well as the data contained in it, is lost when the end of the programming block is reached.

An example program hierarchy:

```
Namespace
     Class
          Field
          Field
          Field
          Method
               Local Variable
               Local Variable
               If X then
                    (block 1)
               Else
                    (block 2)
               End If
          End Method
     End Class
End Namespace
```

Constructor – A special method in a class which executes when a new instance of that class is created. A constructor is used to write the “default” values to the fields in a class. The constructor for a Form is hidden away in the .Designer file for that form.

Property – A special method in a class which is used explicitly to store and retrieve data from a field. If a field must be accessed from outside of the class containing the field, a property must be written for the field, and called upon to store and retrieve the field’s data.

Data type – When you declare that a field or variable will contain a String, the variable is then said to be “of data type String” or “of type String”. Similarly, if you were to declare that a variable will hold an instance of the Shoe class, that variable is said to be “of type Shoe.” The consequence of data typing is that once you declare that a variable will hold one type of data, you cannot later on in the program assign another type of data to this variable. This prevents runtime errors due to an operation on incorrectly formatted data, such as attempting to add a number to a piece of text.

Static (Shared) vs. **Instance** – For a more thorough definition, see pp. 257-266 of the Balena VB book. In summary, fields and methods can be declared either static by using the Shared keyword, or as instance members by omitting the Shared keyword. Static methods cannot access instance fields, and caution should be used when accessing static fields with instance methods. A class cannot be declared Shared. However, the static members (fields and methods) within a particular class are accessible at anytime by any class, without needing to refer to a specific instance of the class created in the program.

Use instance fields to hold all program data that you might need to modify later on.

Form – Called a “window” by the user. In .NET, a form is represented by a class, and that class is split into two files. For a form named Form1, the file Form1.vb holds code written by the programmer. The file Form1.Designer.vb holds code written by Visual Studio which describes the window’s appearance. The .Designer file is not normally accessible from within Visual Studio, and should not be hand-modified by the programmer. Any changes you make to the .Designer file will be overwritten the next time you modify that form’s control surface.

Control – The name for things in the window that the user can click on. They are called “controls” because they allow the user to control program flow and execution. Each control is contained in a class.

3 About the Unit List

The Unit List, labeled `LDUntList` in the source code, is managed by a checkout system. This checkout system consists of the `GetUnit` and `ReturnUnit` methods. When a part of the program needs to access a Unit (an object of type `LDUnit`), it makes a call to `GetUnit`. The calling method can ask `GetUnit` to do one of two things: retrieve the first available unit in the list which has not been downloaded, or retrieve a specific unit according to its ID number (`UnitID` or L-number). If `GetUnit` succeeds in finding the requested unit, it will *remove the unit from* `LDUnitList` and pass it back to the calling method. If `LDUnitList` were a library, `GetUnit`'s function would be to find and check out books to a patron. When the calling method is done with the unit, it returns the Unit to `LDUnitList` by calling `ReturnUnit`. `ReturnUnit` simply adds the Unit to the end of `LDUnitList`.

The checkout system ensures that only one section of the program can access a unit at any one time. This safeguard prevents a unit from being accessed by two modems simultaneously. For example, suppose that modem 1 tries to dial unit 9 at the same time as unit 9 dials into, and connects to, modem 3. Without a checkout system, modem 3 would write valid status information and report data to the Unit that represents unit 9, while modem 1 would write status information to the Unit indicating that the program could not connect to the unit. This possibility of data corruption and/or loss is unacceptable for a program whose mission is to gather data for a research project.

Under the checkout system, either modem 1 or modem 3 would grab exclusive, guaranteed access to the Unit. If modem 1 had access, the call on Modem 3 would be dropped and unit 1 would dial until it reached the (eventually) free modem. If modem 3 grabbed access before modem 1 could, the checkout system would give modem 1 another Unit to call while modem 3 downloaded and stored data from Unit 9.

To maintain the integrity of the queuing system, it was found to be necessary to remove *all* of the Units from `LDUnitList` when the user opened Manual Dial (and later Unit Options), and send them to the Manual Dial (`OneUnitDialog`) or Unit Options (`UnitOptions`) classes. When the user closed either of these windows, the Units were returned, with changes to some of the variables in each Unit, to `LDUnitList`. Taking the units away from other parts of the program ensured synchronization between the data displayed on screed and the data stored in the Units. The downside, of course, is that the four modems cannot obtain any new Units from `LDUnitList`. If a physical unit calls in, it cannot be downloaded. If a modem is already connected to a unit, it can finish that download, but then no more units will be called until Unit Options or Manual Dial are closed.

It was decided to use the checkout mechanism early in the program's development, before becoming familiar with the monitoring *system* as a whole and how the staff at the Aberdeen Test Center (ATC) uses this system. Today there are many situations when it would be advantageous to always keep a complete list of units in memory, and for Unit Options to be able to access a unit at the same time a modem can. It was thought that there would be a need to prevent Unit Options and the modem from writing data to the same variables at once. As it worked out, Unit Options writes to variables that are not written to by the modem, and vice versa. Furthermore, the telephone system permits only one modem to connect to one unit at any given time. Since the download instructions do not execute until the modem is connected to a unit, the rare condition where two modems would try to download a Unit (and overwrite Unit measurement data in the process) is eliminated. It was also thought the program would spend most of its time dialing out to units; it was not anticipated that the program would be used as a server for units calling in. Now the server must be stopped every time the user wants to open Unit Options.

The Unit Viewer is a workaround to not being able to leave Unit Options open indefinitely while the program is running. Unit Viewer gets its data by making a copy of `LDUnitList` and reading the data from the copy. A new copy of `LDUnitList` is sent to Unit Viewer every time `GetUnit` or `ReturnUnit` is called. Unit Viewer compares the new list in memory to the copy of the list stored on disk in order to figure out which units are being downloaded at the moment and display the data appropriately.

Certainly a nondestructive queuing mechanism for `LDUnitList` could be implemented. `GetUnit` could be modified to return the address of the desired Unit on `LDUnitList`, rather than the Unit itself. One would need to use a flag (a Boolean variable) to indicate when a particular unit on the list is being accessed by a modem, so that other units will not grab it. When writing data to a unit, one would need to take care to write only to the *variable* in question, and not attempt to place an entire unit back on the list.

For example, a modem wants to set `IsDownloadInProgress` and `Is-DownloadDone` to False. In the current system, the modem would remove a unit from `LDUnitList`, write to the variables within the unit, then write the entire *unit* back to `LDUnitList`. The code looks something like this:

```
Public Sub Foo() 'some method

   'remove unit 9 from LDUnitList
      Dim unit as LDUnit() = GetUnit("L09")

      'if unit doesn't exist, exit sub before the invalid unit is
      'pushed back onto the LDUnitList
      If unit.UnitNum = "null" Then
            Exit Sub
      End If

      unit.IsDownloadInProgress = False
      unit.IsDownloadDone = False
   ReturnUnit(unit)
End Sub
```

To write the same data without using the checkout system, use this code snippet:

```
Public Sub Foobar() 'another method


   'get the index of unit 9
   Dim idx as Integer = GetUnitIndex("L09")


   'Use Synclock whenever writing to a shared resource, such as
      'LDUnitList. SyncLock is not necessary when reading from the
      'list.


      'Use a Try…Catch block to catch the exception that will be
      'thrown if GetUnitIndex returned an invalid index. Note that
      'this strategy is more flexible than exiting the entire
      'method.


      Try
      Synclock Form1.LDUnitList
         With Form1.LDUnitList.Item(idx)
            .IsDownloadInProgress = False
            .IsDownlaodDone = False
         End With
      End Synclock
   Catch ex As Exception
      'no need to do much here
   End Try
End Sub
```

`GetUnitIndex` is a method similar to `GetUnit`, only instead of returning an actual unit, it returns the unit's position (or address, or index) on `LDUnitList`.

Another issue that remains with nondestructive list management is synchronizing the data displayed in the Manual Dial and Unit Options windows with the current data in `LDUnitList`. An effort will need to be made to keep the function of the Cancel button intact, since it will be convenient to write directly to `LDUnitList` as the user clicks around on the window surface. User Interfaces in general are finicky, and much testing will need to be done to ensure the user can't screw up the download process by changing a unit's options, especially since the wall between Unit Options and the modems will no longer exist.

Perhaps the biggest challenge lies in testing the system. The current list management system has been in use for 6 months, and its strengths and weaknesses are fairly well known. Any new system will need to be tested for many weeks alongside the current system. This will be hard to do without getting extra telephone lines and modems, as testing a queuing system with one modem will not reveal much about that system. It will be a challenge and was not in the scope of this project.

A final note: Whatever queuing system you use, the download and storage of report data will not be affected. Reports are downloaded then immediately checked for errors and written to the SQL database. After the report is written to the database or to disk, the report data are flushed from the Unit, and thus from `LDUnitList`.

4 Time Synchronization

Time synchronization of the units is of critical importance to the noise monitoring system. Without time synchronization, it is impossible to discriminate a blast event from wind noise. Also, it would be impossible to determine an event's origin, which is important for research purposes. L-D Download finds the difference between the clock on each unit and the clock on the computer, and attempts to write a corrected time stamp to each exceedance and interval record using this synchronization data.

Due to the high latency of modem connections, L-D Download does not assume that the 870's clock can be set to the same time as the computer's clock. Instead, L-D Download reads the 870's clock and finds the difference between the 870's clock and the computer's clock. This is referred to as **time offset** in the code. Furthermore, L-D Download tracks the change in the *offset* between the unit and computer over a period of time. This is done to account for drift in the 870's internal clock, and is referred to as **time drift** in the code.

The first step to time synchronization is getting an accurate reading of the unit's clock and determining how much it differs, at that instant, from the host computer's clock. The unit and computer times are read using the `GetUnitTime` function in the Modem class (line 2507 in the Modem.vb file). `GetUnitTime` sends an R2 command to the unit, then grabs the computer's time and stores it to a cell in the `hostDate` array. The critical latency here is the time that passes between when the R2 command is sent and when it is received by the unit. If the computer time is grabbed at the exact second the unit *sends* its response, then the time offset between the computer and unit is accurate. The unit is assumed to send its response immediately after it receives the R2 command. At minimum, the unit will send a response a fixed amount of time after receiving the command; this time can be measured and accounted for by adding the delta to the computer time or else subtracting it from the unit time. Currently, the program adds 45 milliseconds to the stamped computer time to account for a minimum send latency of 45 ms. When the computer receives the response from the unit, it parses the text into a Date object and stores it to a cell in the `unitDate` array. If a response is delayed more than five seconds, the program will not store a response to the `unitDate` array. This error will be caught later on and prompt the program to take more samples of the unit's time.

When using modems, the latency can vary unpredictably, even during the same connection session. When calculating time drift, it is important that a unit that was called on Tuesday have the same latency when called on Wednesday or Thursday, even if the exact latency on Tuesday isn't known. To make the time measurements less susceptible to variations in latency, the unit and computer clocks are polled eight times. For each pair of unit and computer time stamps, the difference (offset) between them is calculated and stored to a cell in the `tDifferenceTemp` array. It is worthy to note that Windows stores a time stamp as a 64-bit integer representing the number of 100-nanosecond intervals that have elapsed since the year 0 A.D. Once the eight unit and computer time stamps are taken, L-D Download computes the numeric average of the stamps by shifting each time stamp three bits to the right (to perform a bitwise division by eight), then adding the shifted stamps together. The bitwise division is used to prevent overflow errors, at the expense of 200 ns' worth of time precision. The averaged unit time is stored in `unitDateTicks` while the averaged computer time is stored in `hostDateTicks`. The average offset is taken by multiplying each of the offsets stored in the `tDifferenceTemp` array by 0.125, then adding them together and storing the result in `tOffsetAvg`. If the time data is valid (no missing time stamps in the `unitDate` and `hostDate` arrays due to a receive timeout), the averaged time data is returned to the calling method as a `ReferenceTime` object.

The second step to time synchronization is determining when each exceedance took place with regards to the computer's clock. To do this, one needs to know two things: the offset between the unit and computer at a previous point in time, and the offset at a time on or after the event took place. For example, suppose that the computer dialed the unit at 0600 and observed that the unit's clock was three seconds ahead of the computer's clock. Suppose that the computer dials the unit again at 1800 hours, only this time observes that the unit is nine seconds ahead of the computer. Assuming that no one tampered with the unit's clock, one may surmise that between 0600 and 1800, the unit's clock was between three and nine seconds ahead of the computer's clock. This means any event that took place at a time between 0600 and 1800, according to the unit, took place between 3 and 9 seconds earlier, according to the computer.

So, when exactly did the event take place? If you can assume that the unit and host clocks drifted in a predictable fashion, you can estimate the offset between the unit and host at the time of the event using offsets taken before and after the event occurred. L-D Download assumes that the unit clock drifts linearly over time, and the computer clock does not drift at all. Using this assumption, if the offsets at a point in time before and after the event are known, then the offset for any event that occurred between these endpoints can be calculated precisely using the equation

$$O_e = O_i + \frac{O_f - O_i}{T_{fh} - T_{ih}} \cdot \left(T_{eu} - \left(T_{ih} + O_i\right)\right) \tag{1}$$

where:

O_e	= offset between unit and host at the time of event
O_i	= offset at a known point in time preceding the event
O_f	= offset at a known point in time following the event
T_{eu}	= time the event occurred, according to the unit
T_{ih}	= computer's time at the known point in time preceding the event
T_{fh}	= computer's time at the known point in time following the event

Putting equation 1 in terms of unit and host times, the time at which the event occurred according to the computer's clock is:

$$H_e = U_e + \left(H_i - U_i\right) + \frac{\left(H_f - U_f\right) - \left(H_i - U_i\right)}{H_f - H_i} \cdot \left(U_e - U_i\right) \tag{2}$$

where:

H_e, U_e	= host (computer) and unit clocks at the time the event occurred
H_i, U_i	= host and unit clocks at the known point in time preceding the event
H_f, U_f	= host and unit clocks at the known point in time following the event

When the unit's time is reset, L-D Download will grab the unit and host times and store them to disk as a serialized ReferenceTime object, using the `SerializeReferenceTime` function in the `LDUnit` class. The program stores one reference time for each unit in the `ReferenceTimes` subfolder of the program folder, using the unit number as the filename. This stored time serves as the known point in time preceding the event; the program will not attempt to append corrected times to any event whose time stamp occurs before this time. The known point in time following the event is the time grabbed when the unit is called, just before a download begins.

The time correction calculation starts at line 1592 in the Modem class file (Modem.vb). For each event and interval read from a unit, L-D Download reads the time stamp set by the unit, then solves equation 2 to determine when the event or interval occurred according to the computer. In the code, H_i and U_i are the variables `RT22.HostTimeZero` and `RT22.UnitTimeZero`, respectively, while H_f and U_f are the local variables `hostDateAvg` and `unitDateAvg`. The difference between the unit and computer event times is O_e from equation 1, which in the code is represented by the block variable drift on line 1619. The word drift is a misnomer, since this variable contains an offset value that is added to the event's time stamp given by the unit. Finally, a corrected time stamp is appended to the end of the event or interval record as a text string.

In a real system, one can assume linear drift if (a) the unit's oscillator (from which the clock is derived) does not change frequency with the temperature variation seen over the course of a typical day, and (b) the computer's clock is continuously synchronized to an stable time source such as the National Institute of Standards and Technology (NIST) Internet time service. Currently, the noise computer is synchronized to NIST once per hour using a software program called NISTIME. Furthermore, Windows Time Service on this machine is disabled; the time service attempted to synchronize the computer to the ATC domain controller, whose clock showed a drift of 1 second per hour during testing in August 2006.

5 Scheduler

When L-D Download is started, a Timer object named `Timer1` is created. The purpose of `Timer1` is to fire an event called Timer1.Tick once each second. A method in Form1 called `Timer1_Tick` handles the Tick event. This `Timer1_Tick` method (line 775 in Form1.vb) and its helper methods `WorkdaySched` and `WeekendSched` (lines 891 and 1065, respectively) form the core of the Scheduler.

Each time Timer1_Tick runs, it grabs the computer time and rounds off the fractional seconds. Next, it looks at the time of day and sets the `Is-Daytime` flag accordingly, then runs either `WorkdaySched` or `Weekend-Sched` depending on the day of the week. `WorkdaySched` and `Weekend-Sched` use a `Select Case` statement to see if it needs to run a scheduled task at this point in time. If the computer's time matches one of times in a `Case` statement, the code beneath that `Case` statement will run. If it does not match a time in the `Case` statement, the code execution will jump to the end of the Select block.

To add an event to the Scheduler, use the following template:

```
'Select Case nowtime 'already in code, nowtime is a TimeSpan
object representing the current computer time

    Case <TimeSpan> 'time at which you want the event to occur,
represented by a TimeSpan object. To schedule an event for 2pm,
you can use "New TimeSpan(14, 0, 0)" or use a class variable that
has an equivalent TimeSpan value.

        Dim uList As List(Of LDUnit) = GetAllUnits() 'grab all
units from LDUnitList

        For Each unit As LDUnit In uList
            'Usually you will have different settings for the
Aberdeen
            'and CERL units. Set the variables for each unit
below.
            'Make sure to set IsDownloadDone and
IsDownloadInProgress
            'to False in order to download the unit.
            If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                unit.AllowCallIns = True
                unit.ResetDataYN = True
                unit.ResetTimeYN = True
                unit.DLTries = 0
                unit.IncludeLC = False
                unit.IncludeQ = False
                unit.IncludeR = False
                unit.IsDownloadDone = False
                unit.IsDownloadInProgress = False
            Else
                unit.AllowCallIns = False
                unit.ResetDataYN = True
                unit.ResetTimeYN = True
                unit.DLTries = 0
                unit.IncludeLC = False
                unit.IncludeQ = False
                unit.IncludeR = False
                unit.IsDownloadDone = False
                unit.IsDownloadInProgress = False
            End If
        'desired exceedance threshold is now determined in the
Modem.Download method
        Next

        ReturnAllUnits(uList) 'return all units to LDUnitList

        'Select which modems to use to call the units - a False
indicates that the modem will dial out, a True indicates that the
modem will be monitoring for incoming calls

        CheckBox1.Checked = False
        CheckBox2.Checked = False
        CheckBox3.Checked = False
        CheckBox4.Checked = False

        'Finally, call the units
        If Not Me.BWDelayedClick.IsBusy Then
            Me.BWDelayedClick.RunWorkerAsync()    'dial out
        Else
            Dim eaM As New
StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingTex
t)
            eaM.PrecedingText = "Cannot start download at " &
Date.Now.ToString("M/d/yyyy h:mm:ss tt") _
            & "due to a busy worker thread." & vbLf & vbLf
            RaiseEvent UpdateMessages(Me, eaM)
        End If

'End Select
```

6 LDUnit Class Variables

The class variables in LDUnit are accessed through their corresponding properties. The variables themselves are marked Protected, while the properties are Public.

```
Public Property InstallPath() As String
```

The base path of L-D Download. By defalut, this is C:\Program Files\L-D Download.

These properties are set by the user, and tell L-D Download how to download the unit.

```
Public Property IncludeR() As Boolean
```

- True – L-D Download will retrieve the Read variables from the unit.
- False – L-D Download will not retrieve the Read variables from the unit.

```
Public Property IncludeLC() As Boolean
```

- True – L-D Download will retrieve the Log and Cal variables from the unit.
- False – L-D Download will not retrieve the Log and Cal variables from the unit.

```
Public Property IncludeQ() As Boolean
```

- True – L-D Download will retrieve the Query variables from the unit.
- False – L-D Download will not retrieve the Query variables from the unit.

```
Public Property AllowCallIns() As Boolean
```

- True – L-D Download will tell the unit to call the noise computer when an exceedance or alarm is recorded (sets Q??? to 3). L-D Download will also enter the noise computer's phone number into the unit.

- False – L-D Download will tell the unit to never call a computer.

```
Public Property ResetDataYN() As Boolean
```

- True – L-D Download will issue a reset-all S1,1 command to the unit. All measurement data on the unit will be erased.
- False – L-D Download will not issue a reset-all command.

```
Public Property ResetTimeYN() As Boolean
```

- True – L-D Download will set the clock on the unit, determine the offset between the unit and computer clocks, and record the offset to disk. Currently the program does not set the date or the day of week on the unit.
- False – L-D Download will not set the clock on the unit.

```
Public Property IsDownloadDone() As Boolean
```

- True if L-D Download completes the download procedure successfully, False if not. Used to determine if a unit needs to be dialed again.
- This variable is only used when a dial command has been issued by the scheduler or user. If a unit calls in and its download fails, that unit will not be re-dialed by the program.

```
Public Property IsDownloadInProgress() As Boolean
```

- True if the unit is currently being downloaded, False otherwise.

```
Public Property IsEnabled() As Boolean
```

- True – When a dial command is issued, L-D Download will dial this unit.
- False – When a dial command is issued, L-D Download will skip this unit.

```
Public Property DLTries() As Integer
```

- After L-D Download has read the exceedance and interval records, it will test to see if the records are in the proper format for entry into the SQL database. If one of the records is not, the program will download all of the records again. DLTries is the number of times the program should attempt to retrieve the E and I records before hanging up the line and declaring the download a failure.

- This property is not set by the user. To change this value, edit the number on line 1475 in Modem.vb and recompile the program.

```
Public Property DialTries() As Integer
```

- The number of times L-D Download will attempt to dial a unit before giving up and declaring the unit was unreachable.

```
Public Property Initialize() As Boolean
```

- True – When a connection with the unit has been established, the `ReconfigureUnit` method will execue. This method first resets the data on a unit, then sets the Q parameters to a default value specified in the code.
- False – When a connection with the unit has been established, the normal `Download` method will execute.

```
Public Property SendCustP() As Boolean
```

- True – Custom parameters will be sent to this unit. The custom parameters are stored in the variable `CustParams`.
- False – Custom parameters will not be sent to this unit.

<u>These properties contain identification data for a unit and are displayed in Unit Options and Unit Viewer.</u>

```
Public Property UnitSerial() As String
```

The serial number of the unit, read from the unit itself.

```
Public Property UnitNum() As String
```

The unit's L number, stored as "L00" through "L99." This is the main unique identifier of a LDUnit object in L-D Download. When the program needs to search for a unit, or sort the list of units, it uses this property as the search criterion.

```
Public Property LockCode() As String
```

The lock code, or password, needed to log onto the unit.

```
Public Property UnitLocation() As String
```

A text string detailing the physical location of the unit. This is what the Aberdeen staff uses to identify a unit.

```
Public Property UnitPhoneNum() As String
```

The telephone number of the unit. When the program wants to dial a unit, it calls this number.

```
Public Property UnitOwner() As Owner
```

The agency that operates the unit. This property is used most often by the program to let the scheduler or user set parameters that are common to just the Aberdeen units, or just the CERL units. For example, the Aberdeen units call in during the day, while the CERL ones do not; the CERL units may have a higher exceedance threshold than the Aberdeen units, etc. The Owner object itself is an enumeration (enum) defined in the LDUnit class. Your choices of owner are:

```
Public Enum Owner
    Aberdeen
    CERL
    Nobody
End Enum
```

```
Public Property LastDL() As DateTime
```

The date and time of the last successful download.

```
Public Property CustParams() As String
```

The custom parameter text for a unit, set through Unit Options.

<u>These properties contain measurement and statistical data for a unit.</u>

```
Public Property NumExceedances() As Integer
```

Number of exceedance records on the unit, as reported by the unit. Used to help verify the completeness of a download, and also displayed in Unit Options and Unit Viewer.

```
Public Property NumIntervals() As Integer
```

Number of interval records on the unit, as reported by the unit.

```
Public Property NumStartStops() As Integer
```

Number of log (L) records on the unit, as reported by the unit.

```
Public Property NumCalibrations() As Integer
```

Number of calibration (C) records on the unit, as reported by the unit.

```
Public Property BattVoltage() As Double
```

The unit's battery voltage.

```
Public Property ErrorString() As String
```

The last eight errors reported by the unit. In Unit Viewer, the program translates the numeric code reported by the unit into a verbose error report.

```
Public Property EList() As List(Of String)
Public Property IList() As List(Of String)
Public Property RList() As List(Of String)
Public Property QList() As List(Of String)
Public Property LList() As List(Of String)
Public Property CList() As List(Of String)
```

These lists store the exceedance, interval, read, query, log, and cal records, respectively, retrieved from the unit. When the program receives the data report, it breaks the report into lines. A line beginning with E is stored to a String in EList, I to IList, and so on. This structuring of the report data aids in checking the report for errors and storing the data to the SQL database or to a text file on disk. Once the report data are checked and written to disk, the data in these lists are deleted to free up memory and to avoid duplicate data being saved to disk.

```
Public Property ExcdThreshold() As Integer
```

The exceedance threshold (in decibels) read from the unit. This value is reported in Unit Options and Unit Viewer.

```
Public Property CalLevel() As Double
```

The calibration level (in decibels) read from the unit. This value is reported in Unit Options and Unit Viewer.

```
Public Property ExcdDay() As Integer
```

The daytime exceedance threshold set by the user.

```
Public Property ExcdNight() As Integer
```

The nighttime exceedance threshold set by the user. If the units are told to stop gathering data altogether at night, this value is not written to the unit.

```
Public Property TimerRun1() As String
```

The time of day at which the unit should start taking data.

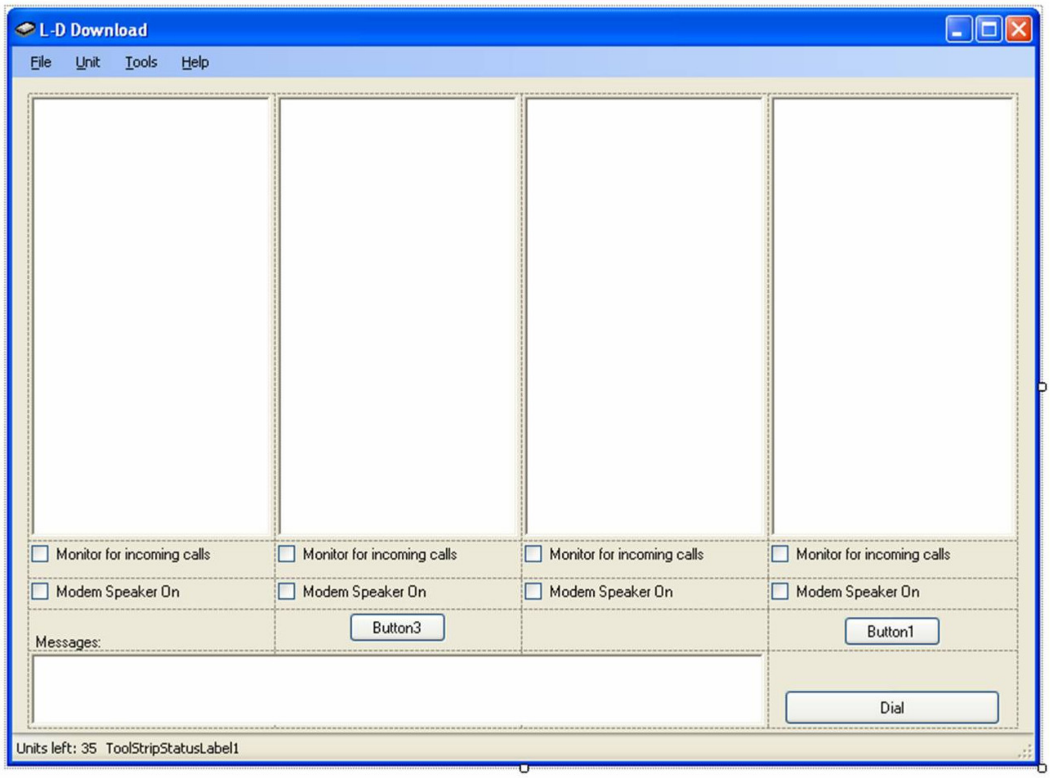
```
Public Property TimerStop1() As String
```

The time of day at which the unit should stop taking data.

```
Public Property TimerChanged() As Boolean
```

- True – The user has changed TimerRun1 or TimerStop1 since the last download. L-D Download will write the new timer settings to the unit.
- False – The user has not changed TimerRun1 or TimerStop1 since the last download. L-D Download will not write new timer settings to the unit.

Appendix A: LD Download (Form1)


L-D Download
File
Unit
Tools
Help
Monitor for incoming calls
Monitor for incoming calls
Monitor for incoming calls
Monitor for incoming calls
Modem Speaker On
Modem Speaker On
Modem Speaker On
Modem Speaker On
Button3
Button1
Messages:
Dial
Units left: 35 ToolStripStatusLabel1

```
'Form1 is the main code it runs the main window.  Everything gets lauchned from here

Option Strict On Compiled code that this code uses

Imports System.Collections.Generic
'Generics - list or array where you don't need to know each index, you can sort through the array get the data.

Imports System.ComponentModel
'Allows you to use VB or .net components. (background worker type) class system

Imports System.Threading
Imports System.Data.SqlClient

Public Class Form1

    'Local variables you need to iniliaze the memory space and for class variables the space is allocated and treated as a null value unit a value is assigned

    Private Shared LDUnitList As New List(Of LDUnit)
    Private WithEvents modem1, modem2, modem3, modem4 As Modem
    Public Shared appset As AppSettings

    'The String = "" initializes a variable and creates a memory address for that variable. This is good practice in coding.  If you don't initialize the declared variable then the code by crash.

    Private Shared currLNum As String = ""  'used by GetUnit
    Private Shared currLNum2 As String = "" 'used by UpdateStatusBoxHandler
    Private wState As FormWindowState

    'Enum is a type of data structure that vb internally uses e.g. 1 = add.  This also enables the code to give you the "n" number of choices when typing your code (e.g. add, create, etc.)

    Private m_closeOK As Boolean
    Private sBarNames As New List(Of String)
    Private sBarPrecedingText As String = ""
    Private sBarText As String = ""
    Private sBarUCount As Integer = 0
    Private stampedTime As TimeSpan
    Private stampedTime2 As TimeSpan  'test!!
    Private wasListening(3) As Boolean
    Private Shared m_isUVopen As Boolean
    Private Shared uv As New UnitView()
    Private Shared m_isDaytime As Boolean
    Private txtOut1h As New RichTextBox()
    Private txtOut2h As New RichTextBox()
    Private txtOut3h As New RichTextBox()
    Private txtOut4h As New RichTextBox()
    'txtoutXh are non-displayed text boxes which hold the text to be written to a log
    'is a workaround for the slowdown that occurs when a RTB has to display a lot of text
    Private rtfBody1 As New List(Of String)
    Private rtfBody2 As New List(Of String)
    Private rtfBody3 As New List(Of String)
    Private rtfBody4 As New List(Of String)
    Private header As String = "{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fnil\fcharset0 Microsoft Sans Serif;}}" & vbLf & "{\colortbl ;\red0\green128\blue0;\red0\green0\blue0;\red255\green0\blue0;}" & vbLf & _
   "\viewkind4\uc1\pard\f0\fs17"
    Private footer As String = "\cf0\par" & vbLf & "}" & vbLf

    Public Event SBUnitCounter As EventHandler(Of StatusBarEventArgs)
    Public Event UpdateMessages As EventHandler(Of StatusBarEventArgs)


    Public ReadOnly Property ReadLDUnitList() As List(Of LDUnit)
        Get
            Dim ulist As New List(Of LDUnit)
            ulist.AddRange(LDUnitList)
            Return ulist
        End Get
    End Property
    Public Shared Property IsUVOpen() As Boolean
        Get
            Return m_isUVopen
        End Get
        Set(ByVal value As Boolean)
            m_isUVopen = value
        End Set
    End Property
    Public Shared Property IsDaytime() As Boolean
        Get
            Return m_isDaytime
        End Get
        Set(ByVal value As Boolean)
            m_isDaytime = value
        End Set
    End Property
    'IMPORTANT: To read from the list, then write back to the list,
    'use the GetAllUnitsNoMark/ReturnAllUnits mechanism and let garbage collection handle the copied uList.
    'Do NOT write a property - overwriting LDUnitList many times in succession led to lost and/or duplicate
    'units


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Button1.Hide()
        Button3.Hide()
        Me.lblStatus.Text = ""
        appset = New AppSettings()

        Try
            Using fs23 As New FileStream(appset.InstallPath & "app.dat", FileMode.Open)
                'New declares a new object or class
                'appset.InstallPath is C:\Program Files\LD_Download"

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                appset = DirectCast(bf.Deserialize(fs23), AppSettings)

            End Using
        Catch ex As FileNotFoundException   'file does not exist - load default settings
            appset = New AppSettings()
        End Try

        'routines to instantiate Modem and LDUnit objects
        modem1 = New Modem(appset.Modem1Port, appset.Modem1Num, Modem.Mode.Standby)
        modem2 = New Modem(appset.Modem2Port, appset.Modem2Num, Modem.Mode.Standby)
        modem3 = New Modem(appset.Modem3Port, appset.Modem3Num, Modem.Mode.Standby)
        modem4 = New Modem(appset.Modem4Port, appset.Modem4Num, Modem.Mode.Standby)

        AddHandler modem1.UpdateOutBoxesEvent, AddressOf UpdateTxtOut1Handler
        AddHandler modem2.UpdateOutBoxesEvent, AddressOf UpdateTxtOut2Handler
        AddHandler modem3.UpdateOutBoxesEvent, AddressOf UpdateTxtOut3Handler
        AddHandler modem4.UpdateOutBoxesEvent, AddressOf UpdateTxtOut4Handler

        'Handlers will be removed automatically when application closes

        Try

            'If there is a problem with units being taken away from the list or more units being added ↙
    to the list, the you can delete the units.dat and it will load in the defaults which are hard coded ↙
    below

            Using fs As New FileStream(appset.InstallPath & "units.dat", FileMode.Open)

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                LDUnitList = DirectCast(bf.Deserialize(fs), List(Of LDUnit))

            End Using
        Catch ex As FileNotFoundException   'file does not exist
            'Hard-code unit identifiers and phone numbers, for now
            'units that are safe-ish to test
            'LDUnitList.Add(New LDUnit("L38", "912176201877", LDUnit.Owner.CERL))
            LDUnitList.Add(New LDUnit("L06", "14102737782", LDUnit.Owner.Aberdeen, "Perryman"))
            LDUnitList.Add(New LDUnit("L07", "14106766443", LDUnit.Owner.Aberdeen, "Long Bar Harbor"))
            LDUnitList.Add(New LDUnit("L08", "14107789023", LDUnit.Owner.Aberdeen, "Tolchester"))
            LDUnitList.Add(New LDUnit("L13", "14109395258", LDUnit.Owner.Aberdeen, "Havre de Grace"))
            LDUnitList.Add(New LDUnit("L15", "14103355733", LDUnit.Owner.Aberdeen, "Chase"))

            'remaining APG units
            LDUnitList.Add(New LDUnit("L01", "14107789024", LDUnit.Owner.Aberdeen, "Stoops Point"))
            LDUnitList.Add(New LDUnit("L03", "14108100350", LDUnit.Owner.Aberdeen, "Howell Point"))
            LDUnitList.Add(New LDUnit("L04", "14102751287", LDUnit.Owner.Aberdeen, "Grove Point"))
            LDUnitList.Add(New LDUnit("L05", "14102721688", LDUnit.Owner.Aberdeen, "B Tower"))
            LDUnitList.Add(New LDUnit("L09", "14107789025", LDUnit.Owner.Aberdeen, "Worton Point"))
            LDUnitList.Add(New LDUnit("L10", "14102752384", LDUnit.Owner.Aberdeen, "Crystal Beach"))
            LDUnitList.Add(New LDUnit("L11", "14103351960", LDUnit.Owner.Aberdeen, "Bowley's Quarter"))
            LDUnitList.Add(New LDUnit("L12", "14103482211", LDUnit.Owner.Aberdeen, "Betterton"))
            LDUnitList.Add(New LDUnit("L14", "14102737529", LDUnit.Owner.Aberdeen, "Aberdeen"))
            LDUnitList.Add(New LDUnit("L16", "14102878283", LDUnit.Owner.Aberdeen, "Red Point"))
            LDUnitList.Add(New LDUnit("L17", "14102878401", LDUnit.Owner.Aberdeen, "Charlestown"))
            LDUnitList.Add(New LDUnit("L18", "14106761631", LDUnit.Owner.Aberdeen, "Edgewood"))
            LDUnitList.Add(New LDUnit("L19", "14102870492", LDUnit.Owner.Aberdeen, "Turkey Point"))

            'CERL units
            LDUnitList.Add(New LDUnit("L21", "12175217254", LDUnit.Owner.CERL, "Sparrows Point"))
            LDUnitList.Add(New LDUnit("L22", "12175218691", LDUnit.Owner.CERL, "Essex"))
            LDUnitList.Add(New LDUnit("L23", "12176215594", LDUnit.Owner.CERL, "Baltimore"))
            LDUnitList.Add(New LDUnit("L24", "12175218692", LDUnit.Owner.CERL, "Fork"))
            LDUnitList.Add(New LDUnit("L25", "12175218693", LDUnit.Owner.CERL, "Bel Air"))
            LDUnitList.Add(New LDUnit("L26", "12176201877", LDUnit.Owner.CERL, "Churchville"))
            LDUnitList.Add(New LDUnit("L27", "12175218695", LDUnit.Owner.CERL, "Darlington"))
            LDUnitList.Add(New LDUnit("L28", "12175218696", LDUnit.Owner.CERL, "Port Deposit"))
            LDUnitList.Add(New LDUnit("L29", "12175218697", LDUnit.Owner.CERL, "Abingdon"))
            LDUnitList.Add(New LDUnit("L30", "12175218698", LDUnit.Owner.CERL, "North East"))
            LDUnitList.Add(New LDUnit("L31", "12175218800", LDUnit.Owner.CERL, "Elkton"))
            LDUnitList.Add(New LDUnit("L32", "12175218802", LDUnit.Owner.CERL, "Chesapeake City"))
            LDUnitList.Add(New LDUnit("L33", "12175218803", LDUnit.Owner.CERL, "Cecilton"))
            LDUnitList.Add(New LDUnit("L34", "12175218807", LDUnit.Owner.CERL, "Millington"))
            LDUnitList.Add(New LDUnit("L35", "12175218808", LDUnit.Owner.CERL, "Still Pond"))
            LDUnitList.Add(New LDUnit("L36", "12175218809", LDUnit.Owner.CERL, "Chestertown"))
            LDUnitList.Add(New LDUnit("L37", "12175218810", LDUnit.Owner.CERL, "Rock Hall"))

        End Try

        MakeUnitsDownloadable()
```

```vb

       'The Me. accesses all the methods and variables available to you and allows you to select from those available.  It keeps you from having to type in the variable names over and over.  It also distiguishes between a class variable in that class and a local variable in that method.

        'The Me. keyword more specifically denotes an instance class variable - that is, a new copy of this variable is created for each instance of a class in a program. If one instance of a class changes the instance variable "dollar", the other instances will not have their "dollar" variables changes as well.

        'This is different for a 'Shared' variable, which is a class variable that is common to all instances of a class. If one instance of a class chages the data in a shared variable, the change will be visible to all other instances of that class. Shared variables are not accessible using the Me keyword.

        'other program settings
        NotifyIcon1.Visible = False
        Me.wState = Me.WindowState
        Me.m_closeOK = False
        Dim stampsec As Long = CLng(Date.Now.TimeOfDay.TotalSeconds)
        Me.stampedTime = New TimeSpan(stampsec * 10000000)
        'Time is stamped here so it can be used later in the process of determining when 25 minutes has elapsed.
        IsUVOpen = False
        IsDaytime = False

        'test!!
        Dim stamphr As Long = CLng(Date.Now.TimeOfDay.Hours)  'test - rounds time down to the nearest hour
        Me.stampedTime2 = New TimeSpan(stamphr * 60 * 60 * 10000000)
        Me.stampedTime2 = Me.stampedTime2.Subtract(New TimeSpan(0, 30, 0))
        'end test!!

        Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Clear)
        RaiseEvent SBUnitCounter(Me, EA)
        RaiseEvent UpdateMessages(Me, EA)
        Me.lblStatus.Text = ""

    End Sub
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

        'Set this up as a test button.  It is not used in the program.

        'test the date parsing function
        Dim unitTime As String = "Wed 28Jun2006 13:10:15"
        Dim hostdate As Date = Date.Now
        Dim tOffset As Integer = 0
        Dim tOffsetTemp As TimeSpan = New TimeSpan(0)
        Dim tOffsetTemp2 As Double = -1
        Dim unitdate As Date = New Date(1)
        Dim hostDTTicks As Long = 0

        If Not unitTime Like "#*" Then
            unitTime = unitTime.Remove(0, 4)    'trim off the day of week from the date - it is redundant
            '                                   and will introduce errors if it was set wrong on the 870
        End If

        unitdate = Date.Parse(unitTime)
        unitdate = unitdate.AddMilliseconds(45)

        For i As Integer = 0 To 7
            hostdate = Date.Now
            hostDTTicks += hostdate.Ticks >> 3
            Threading.Thread.Sleep(125)
        Next

        Dim uuu As Date = New Date(hostDTTicks)

        tOffsetTemp = hostdate.Subtract(unitdate)
        tOffsetTemp2 = tOffsetTemp.TotalSeconds Mod 3600    'ignore the difference in hours -
        '                                                       prevents time zone differences from
        '                                                       screwing up the recorded time

        tOffset = CInt(Math.Round(tOffsetTemp2, 0, MidpointRounding.AwayFromZero))

        'MessageBox.Show(unitdate.ToString & vbCr & hostdate.ToString & vbCr & tOffsetTemp.ToString & vbCr & tOffsetTemp2.ToString & vbCr & tOffset.ToString & vbCr & uuu.ToString, "Unit time, Host time, offset, offset seconds only")

        MessageBox.Show(hostdate.Hour & ":" & hostdate.Minute & ":" & hostdate.Second)

        'Dim ppp As Integer = Date.Now.DayOfWeek + 1
        'MessageBox.Show(ppp.ToString)

        'Dim conn As New SqlClient.SqlConnection("server=localhost; database=Complaint; Integrated Security=SSPI")
        'Dim cmd As New SqlCommand("", conn)

        'Try
        '    conn.Open()
        '    cmd.CommandText = "SELECT count(UnitID) FROM Intervals"
        '    Dim numIRows As Integer = CInt(cmd.ExecuteScalar())
        '    For i As Integer = 1 To numIRows
        '        cmd.CommandText = "Select Corrected_Time from Intervals where recordnum = " & i.ToString
        '        Try
        '            Dim dt3 As Date = DirectCast(cmd.ExecuteScalar, DateTime)
        '            Dim dt4 As New TimeSpan(dt3.Ticks)
        '            Dim hrs As Double = dt4.TotalHours
        '            hrs = Math.Round(hrs, 2)
        '            cmd.CommandText = "update Intervals set TotalHrs_Corrected = " & hrs.ToString & " where ↙
     recordnum = " & i.ToString
        '            cmd.ExecuteScalar()
        '        Catch ex As Exception
        '            'don't update the db
        '        End Try
        '        Application.DoEvents()
        '    Next


        '    cmd.CommandText = "SELECT count(UnitID) FROM EventData"
        '    Dim numERows As Integer = CInt(cmd.ExecuteScalar())
        '    For i As Integer = 1 To numERows
        '        cmd.CommandText = "Select Corrected_Time from EventData where recordnum = " & i.ToString
        '        Try
        '            Dim dt3 As Date = DirectCast(cmd.ExecuteScalar, DateTime)
        '            Dim dt4 As New TimeSpan(dt3.Ticks)
        '            Dim hrs As Double = dt4.TotalHours
        '            hrs = Math.Round(hrs, 2)
        '            cmd.CommandText = "update EventData set TotalHrs_Corrected = " & hrs.ToString & " where ↙
     recordnum = " & i.ToString
        '            cmd.ExecuteScalar()
        '        Catch ex As Exception
        '            'don't update the db
        '        End Try
        '        Application.DoEvents()
        '    Next

        '    cmd.CommandText = "SELECT count(UnitID) FROM EventData"
        '    numERows = CInt(cmd.ExecuteScalar())
        '    For i As Integer = 1 To numERows
        '        cmd.CommandText = "Select Date_Time from EventData where recordnum = " & i.ToString
        '        Try
        '            Dim dt3 As Date = DirectCast(cmd.ExecuteScalar, DateTime)
        '            Dim dt4 As New TimeSpan(dt3.Ticks)
        '            Dim hrs As Double = dt4.TotalHours
        '            hrs = Math.Round(hrs, 2)
        '            cmd.CommandText = "update EventData set TotalHrs = " & hrs.ToString & " where recordnum↙
     = " & i.ToString
        '            cmd.ExecuteScalar()
        '        Catch ex As Exception
        '            'don't update the db
        '        End Try
        '        Application.DoEvents()
        '    Next
        'Catch ex As SqlException
        '    MessageBox.Show(ex.Message)
        'Finally
        '    conn.Close()
        'End Try


    End Sub 'NOT USED IN THE PROGRAM


     'The #Region is not used by the program, it is just a way to group a region of code together to make   ↙
     it easier to read


#Region "text box handlers"
    'These methods run on separate threads, since they are event handlers,
    'and therefore need delegates to point to methods which actually update the textboxes

    'These Event handlers send text to the textboxes.

    Public Sub UpdateTxtOut1Handler(ByVal sender As Object, ByVal e As UpdateOutBoxesEventArgs)
        Dim deleg As UpdateTxtOutDelegate
        Dim obj As Object() = {e}
        deleg = New UpdateTxtOutDelegate(AddressOf UpdateTxtOut1)
        txtOut1.Invoke(deleg, obj)
    End Sub
    Public Sub UpdateTxtOut2Handler(ByVal sender As Object, ByVal e As UpdateOutBoxesEventArgs)
        Dim deleg As UpdateTxtOutDelegate
        Dim obj As Object() = {e}
        deleg = New UpdateTxtOutDelegate(AddressOf UpdateTxtOut2)
        txtOut1.Invoke(deleg, obj)
    End Sub
    Public Sub UpdateTxtOut3Handler(ByVal sender As Object, ByVal e As UpdateOutBoxesEventArgs)
        Dim deleg As UpdateTxtOutDelegate
        Dim obj As Object() = {e}
        deleg = New UpdateTxtOutDelegate(AddressOf UpdateTxtOut3)
        txtOut3.Invoke(deleg, obj)
    End Sub
    Public Sub UpdateTxtOut4Handler(ByVal sender As Object, ByVal e As UpdateOutBoxesEventArgs)
        Dim deleg As UpdateTxtOutDelegate
        Dim obj As Object() = {e}
        deleg = New UpdateTxtOutDelegate(AddressOf UpdateTxtOut4)
        txtOut4.Invoke(deleg, obj)
    End Sub

    'Delegate declaration and worker methods for updating the text boxes
    'These worker methods run on the UI thread
    Public Delegate Sub UpdateTxtOutDelegate(ByVal e As UpdateOutBoxesEventArgs)
    Public Sub UpdateTxtOut1(ByVal e As UpdateOutBoxesEventArgs)
        Dim textColor As Color
        If e.RWFlag = UpdateOutBoxesEventArgs.RW.Read Then
            textColor = Color.Red
        ElseIf e.RWFlag = UpdateOutBoxesEventArgs.RW.Write Then
            textColor = Color.Black
        Else
            textColor = Color.Green
        End If
        With txtOut1h 'write a copy to the hidden box
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
        End With

        'ghetto fix
        With txtOut1
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
            .ScrollToCaret()
        End With
        If txtOut1.Lines.GetLength(0) > appset.MaxLines Then 'clear - yes, it's a ghetto fix
            txtOut1.Clear()
        End If
        'end ghetto fix

        ''generate RTF code
        'Dim rtfStr As New System.Text.StringBuilder()
        'Dim colorCtrl As String = ""

        'Select Case e.RWFlag
        '    Case UpdateOutBoxesEventArgs.RW.Notify : colorCtrl = vbLf & "\cf1 "
        '    Case UpdateOutBoxesEventArgs.RW.Write : colorCtrl = vbLf & "\cf2 "
        '    Case UpdateOutBoxesEventArgs.RW.Read : colorCtrl = vbLf & "\cf3 "
        'End Select

        'Dim echars() As Char = e.AppendThisString.ToCharArray()
        'For Each c As Char In echars
        '    If Microsoft.VisualBasic.AscW(c) > 127 Then
        '        c = "_"c
        '    End If
        'Next
        'e.AppendThisString = String.Concat(echars)
        'e.AppendThisString = e.AppendThisString.Replace(vbLf, "\par ")
        'e.AppendThisString = e.AppendThisString.Replace(vbCr, "\par ")

        'Me.rtfBody1.Add(colorCtrl & e.AppendThisString)
        'If Me.rtfBody1.Count > appset.MaxLines Then 'remove the oldest line(s)
        '    Do
        '        Me.rtfBody1.RemoveAt(0)
        '    Loop Until Me.rtfBody1.Count <= appset.MaxLines
        'End If

        'For i As Integer = 0 To Me.rtfBody1.Count - 1
        '    rtfStr.Append(Me.rtfBody1.Item(i))
        'Next


        ''post the rtf string to the screen
        'txtOut1.SuspendLayout()
        'With txtOut1
        '    .Rtf = header & rtfStr.ToString & footer
        '    .Select(Me.txtOut1.TextLength, 0)
        '    If Not .Focused Then
        '        .ScrollToCaret()
        '        'if txtout1 has the focus, it will automatically scroll to the end of the selection
        '    End If
        'End With
        'txtOut1.ResumeLayout()

        '        'Me.Text = rtfBody1.Count.ToString & "   " & rtfBody2.Count.ToString & "   " & _
        '        '  rtfBody3.Count.ToString & "   " & rtfBody4.Count.ToString
    End Sub
    Public Sub UpdateTxtOut2(ByVal e As UpdateOutBoxesEventArgs)
        Dim textColor As Color
        If e.RWFlag = UpdateOutBoxesEventArgs.RW.Read Then
            textColor = Color.Red
        ElseIf e.RWFlag = UpdateOutBoxesEventArgs.RW.Write Then
            textColor = Color.Black
        Else
            textColor = Color.Green
        End If
        With txtOut2h 'write a copy to the hidden box
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
        End With

        'ghetto fix
        With txtOut2
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
            .ScrollToCaret()
        End With
        If txtOut2.Lines.GetLength(0) > appset.MaxLines Then 'clear - yes, it's a ghetto fix
            txtOut2.Clear()
        End If
        'end ghetto fix

        ''generate RTF code
        'Dim rtfStr As New System.Text.StringBuilder()
        'Dim colorCtrl As String = ""

        'Select Case e.RWFlag
        '    Case UpdateOutBoxesEventArgs.RW.Notify : colorCtrl = vbLf & "\cf1 "
        '    Case UpdateOutBoxesEventArgs.RW.Write : colorCtrl = vbLf & "\cf2 "
        '    Case UpdateOutBoxesEventArgs.RW.Read : colorCtrl = vbLf & "\cf3 "
        'End Select

        'Dim echars() As Char = e.AppendThisString.ToCharArray()
        'For Each c As Char In echars
        '    If Microsoft.VisualBasic.AscW(c) > 127 Then
        '        c = "_"c
        '    End If
        'Next
        'e.AppendThisString = String.Concat(echars)
        'e.AppendThisString = e.AppendThisString.Replace(vbLf, "\par ")
        'e.AppendThisString = e.AppendThisString.Replace(vbCr, "\par ")

        'Me.rtfBody2.Add(colorCtrl & e.AppendThisString)
        'If Me.rtfBody2.Count > appset.MaxLines Then 'remove the oldest line(s)
        '    Do
        '        Me.rtfBody2.RemoveAt(0)
        '    Loop Until Me.rtfBody2.Count <= appset.MaxLines
        'End If

        'For i As Integer = 0 To Me.rtfBody2.Count - 1
        '    rtfStr.Append(Me.rtfBody2.Item(i))
        'Next


        ''post the rtf string to the screen
        'txtOut2.SuspendLayout()
        'With txtOut2
        '    .Rtf = header & rtfStr.ToString & footer
        '    .Select(Me.txtOut2.TextLength, 0)
        '    If Not .Focused Then
        '        .ScrollToCaret()
        '    End If
        'End With
        'txtOut2.ResumeLayout()

        ''         Me.Text = rtfBody1.Count.ToString & "   " & rtfBody2.Count.ToString & "   " & _
        ''         rtfBody3.Count.ToString & "   " & rtfBody4.Count.ToString
    End Sub
    Public Sub UpdateTxtOut3(ByVal e As UpdateOutBoxesEventArgs)
        Dim textColor As Color
        If e.RWFlag = UpdateOutBoxesEventArgs.RW.Read Then
            textColor = Color.Red
        ElseIf e.RWFlag = UpdateOutBoxesEventArgs.RW.Write Then
            textColor = Color.Black
        Else
            textColor = Color.Green
        End If
        With txtOut3h 'write a copy to the hidden box
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
        End With

        'ghetto fix
        With txtOut3
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
            .ScrollToCaret()
        End With
        If txtOut3.Lines.GetLength(0) > appset.MaxLines Then 'clear - yes, it's a ghetto fix
            txtOut3.Clear()
        End If
        'end ghetto fix


        ''generate RTF code
        'Dim rtfStr As New System.Text.StringBuilder()
        'Dim colorCtrl As String = ""

        'Select Case e.RWFlag
        '    Case UpdateOutBoxesEventArgs.RW.Notify : colorCtrl = vbLf & "\cf1 "
        '    Case UpdateOutBoxesEventArgs.RW.Write : colorCtrl = vbLf & "\cf2 "
        '    Case UpdateOutBoxesEventArgs.RW.Read : colorCtrl = vbLf & "\cf3 "
        'End Select

        'Dim echars() As Char = e.AppendThisString.ToCharArray()
        'For Each c As Char In echars
        '    If Microsoft.VisualBasic.AscW(c) > 126 OrElse Microsoft.VisualBasic.AscW(c) < 9 OrElse _
        '    Microsoft.VisualBasic.AscW(c) = 123 OrElse Microsoft.VisualBasic.AscW(c) = 125 OrElse _
        '    Microsoft.VisualBasic.AscW(c) = 92 Then
        '        c = "_"c
        '    End If
        'Next
        'e.AppendThisString = String.Concat(echars)
        'e.AppendThisString = e.AppendThisString.Replace(vbLf, "\par ")
        'e.AppendThisString = e.AppendThisString.Replace(vbCr, "\par ")

        'Me.rtfBody3.Add(colorCtrl & e.AppendThisString)
        'If Me.rtfBody3.Count > appset.MaxLines Then 'remove the oldest line(s)
        '    Do
        '        Me.rtfBody3.RemoveAt(0)
        '    Loop Until Me.rtfBody3.Count <= appset.MaxLines
        'End If

        'For i As Integer = 0 To Me.rtfBody3.Count - 1
        '    rtfStr.Append(Me.rtfBody3.Item(i))
        'Next

        ''Me.rtbMessages.AppendText(Me.rtfBody3.Item(Me.rtfBody3.Count - 1))

        ''post the rtf string to the screen
        'txtOut3.SuspendLayout()
        'With txtOut3
        '    .Rtf = header & rtfStr.ToString & footer
        '    .Select(Me.txtOut3.TextLength, 0)
        '    If Not .Focused Then
        '        .ScrollToCaret()
        '    End If
        'End With
        'txtOut3.ResumeLayout()

        '        'Me.Text = rtfBody1.Count.ToString & "   " & rtfBody2.Count.ToString & "   " & _
        '        'rtfBody3.Count.ToString & "   " & rtfBody4.Count.ToString
    End Sub
    Public Sub UpdateTxtOut4(ByVal e As UpdateOutBoxesEventArgs)
        Dim textColor As Color
        If e.RWFlag = UpdateOutBoxesEventArgs.RW.Read Then
            textColor = Color.Red
        ElseIf e.RWFlag = UpdateOutBoxesEventArgs.RW.Write Then
            textColor = Color.Black
        Else
            textColor = Color.Green
        End If
        With txtOut4h 'write a copy to the hidden box
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
        End With

        'ghetto fix
        With txtOut4
            .SelectionColor = textColor
            .AppendText(e.AppendThisString)
            .ScrollToCaret()
        End With
        If txtOut4.Lines.GetLength(0) > appset.MaxLines Then 'clear - yes, it's a ghetto fix
            txtOut4.Clear()
        End If
        'end ghetto fix


        ''generate RTF code
        'Dim rtfStr As New System.Text.StringBuilder()
        'Dim colorCtrl As String = ""

        'Select Case e.RWFlag
        '    Case UpdateOutBoxesEventArgs.RW.Notify : colorCtrl = vbLf & "\cf1 "
        '    Case UpdateOutBoxesEventArgs.RW.Write : colorCtrl = vbLf & "\cf2 "
        '    Case UpdateOutBoxesEventArgs.RW.Read : colorCtrl = vbLf & "\cf3 "
        'End Select

        'Dim echars() As Char = e.AppendThisString.ToCharArray()
        'For Each c As Char In echars
        '    If Microsoft.VisualBasic.AscW(c) > 127 Then
        '        c = "_"c
        '    End If
        'Next
        'e.AppendThisString = String.Concat(echars)
        'e.AppendThisString = e.AppendThisString.Replace(vbLf, "\par ")
        'e.AppendThisString = e.AppendThisString.Replace(vbCr, "\par ")

        'Me.rtfBody4.Add(colorCtrl & e.AppendThisString)
        'If Me.rtfBody4.Count > appset.MaxLines Then 'remove the oldest line(s)
        '    Do
        '        Me.rtfBody4.RemoveAt(0)
        '    Loop Until Me.rtfBody4.Count <= appset.MaxLines
        'End If

        'For i As Integer = 0 To Me.rtfBody4.Count - 1
        '    rtfStr.Append(Me.rtfBody4.Item(i))
        'Next


        ''post the rtf string to the screen
        'txtOut4.SuspendLayout()
        'With txtOut4
        '    .Rtf = header & rtfStr.ToString & footer
        '    .Select(Me.txtOut4.TextLength, 0)
        '    If Not .Focused Then
        '        .ScrollToCaret()
        '    End If
        'End With
        'txtOut4.ResumeLayout()

        '        'Me.Text = rtfBody1.Count.ToString & "   " & rtfBody2.Count.ToString & "   " & _
        '        '   rtfBody3.Count.ToString & "   " & rtfBody4.Count.ToString
    End Sub

#End Region

#Region "Status Bar Methods"

    Public Sub SBUnitCounterHandler(ByVal sender As Object, ByVal e As StatusBarEventArgs) Handles Me.    ↙
    SBUnitCounter, modem1.SBUnitCounter, modem2.SBUnitCounter, modem3.SBUnitCounter, modem4.SBUnitCounter

        Dim labeltext As String = lblCounter.Text

        Select Case e.Action
            Case StatusBarEventArgs.SBAction.Populate
                'finds total # of units to be downloaded:
                Me.sBarUCount = 0
                For Each unit As LDUnit In LDUnitList
                    If unit.IsDownloadDone = False AndAlso unit.IsDownloadInProgress = False Then
                        Me.sBarUCount += 1
                    End If
                Next
                labeltext = "Units left: " & Me.sBarUCount.ToString & "   "
            Case StatusBarEventArgs.SBAction.Add
                Me.sBarUCount += 1
                labeltext = "Units left: " & Me.sBarUCount.ToString & "   "
            Case StatusBarEventArgs.SBAction.Remove
                Me.sBarUCount -= 1
                If Me.sBarUCount < 0 Then
                    Me.sBarUCount = 0
                End If
                labeltext = "Units left: " & Me.sBarUCount.ToString & "   "
            Case StatusBarEventArgs.SBAction.Clear
                Me.sBarUCount = 0
                labeltext = ""
            Case StatusBarEventArgs.SBAction.Cancelled
            Case StatusBarEventArgs.SBAction.ChangePrecedingText
            Case StatusBarEventArgs.SBAction.HideText
                labeltext = ""
            Case StatusBarEventArgs.SBAction.ShowText
                labeltext = "Units left: " & Me.sBarUCount.ToString & "   "
        End Select

        lblCounter.Text = labeltext
        uv.tsslUnitsLeft.Text = "    " & labeltext

    End Sub

    Private Sub Form1_UpdateMessages(ByVal sender As Object, ByVal e As StatusBarEventArgs) Handles Me.    ↙
    UpdateMessages, modem1.UpdateMessages, modem2.UpdateMessages, modem3.UpdateMessages, modem4.      ↙
    UpdateMessages
        'updates the message box
        'Populate mode writes "Dial sequence started at (date) (time)" and populates the (invisible) unit ↙
    list
        'Add, Remove add and remove units
        'ShowText runs when a dial sequence is finished and displays
        ' "Dial sequence complete at (date) (time)"
        ' "Units (units) did not download."

        Dim unitString As New System.Text.StringBuilder
        Dim trimchars() As Char = {"L"c, "0"c}

        Select Case e.Action
            Case StatusBarEventArgs.SBAction.Populate
                'clears the list, then populates it with units that are about to be downloaded
                Me.sBarNames.Clear()
                SyncLock LDUnitList
                    For Each unit As LDUnit In LDUnitList
                        If unit.IsDownloadDone = False AndAlso unit.IsDownloadInProgress = False Then
                            Me.sBarNames.Add(unit.UnitNum & "  ")
                        End If
                    Next
                End SyncLock
                Me.sBarNames.Sort()
                For Each str As String In Me.sBarNames
                    unitString.Append(str.TrimStart(trimchars))
                Next
                Me.rtbMessages.AppendText("Dial sequence started " & Date.Now.ToString("M/d/yyyy h:mm:ss tt"↙
    ) _
                & "." & vbLf)
                Me.rtbMessages.Select(Me.rtbMessages.TextLength, 0) 'moves caret to end of text box
                Me.rtbMessages.ScrollToCaret()
            Case StatusBarEventArgs.SBAction.Add
                'add a unit name to SbarNames and display it
                Me.sBarNames.Add(e.UnitNum & "  ")
                Me.sBarNames.Sort()
            Case StatusBarEventArgs.SBAction.Remove
                'remove a unit name and update the display
                currLNum2 = e.UnitNum
                Dim idx As Integer = Me.sBarNames.FindIndex(AddressOf IsUnitName)
                If idx > -1 Then
                    Me.sBarNames.RemoveAt(idx)
                End If
                Me.sBarNames.Sort()
            Case StatusBarEventArgs.SBAction.Clear
                Me.sBarNames.Clear()
                Me.sBarPrecedingText = ""
                Me.rtbMessages.Clear()
            Case StatusBarEventArgs.SBAction.Cancelled
                Me.rtbMessages.AppendText("Dial sequence canceled " & Date.Now.ToString("M/d/yyyy h:mm:ss tt↙
    ") _
                & "." & vbLf)
                Me.rtbMessages.Select(Me.rtbMessages.TextLength, 0) 'moves caret to end of text box
                Me.rtbMessages.ScrollToCaret()
            Case StatusBarEventArgs.SBAction.ChangePrecedingText
                'Add a line of text to the message box
                Me.rtbMessages.AppendText(e.PrecedingText)
            Case StatusBarEventArgs.SBAction.HideText
            Case StatusBarEventArgs.SBAction.ShowText
                'displays date and time download completes
                Me.sBarNames.Sort()
                For Each str As String In Me.sBarNames
                    unitString.Append(str.TrimStart(trimchars))
                Next
                Me.rtbMessages.AppendText("Dial sequence completed " & Date.Now.ToString("M/d/yyyy h:mm:ss  ↙
    tt") _
                & "." & vbLf)
                If Me.sBarNames.Count = 0 Then
                    Me.rtbMessages.AppendText("All units downloaded successfully." & vbLf & vbLf)
                Else
                    Me.rtbMessages.AppendText("Units " & unitString.ToString & "did not download." & vbLf & ↙
    vbLf)
                    'uv.tsslUnitsLeft.Text = "Units " & unitString.ToString & "did not download."
                End If
                Me.rtbMessages.Select(Me.rtbMessages.TextLength, 0) 'moves caret to end of text box
                Me.rtbMessages.ScrollToCaret()

        End Select
    End Sub


    Private Shared Function IsUnitName(ByVal str As String) As Boolean
        If String.Compare(str, currLNum2 & "  ") = 0 Then
            Return True
        Else
            Return False
        End If
    End Function


#End Region

#Region "Scheduler"

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick

        Dim nightlyDL1 As New TimeSpan(21, 0, 0)    'use for test only
        Dim nightlyDL2 As New TimeSpan(22, 15, 0)   'use for test only

        Dim now As Date = Date.Now
        Dim nowDay As DayOfWeek = now.DayOfWeek
        Dim nowTime As TimeSpan = now.TimeOfDay
        Dim diffTime As TimeSpan = nowTime.Subtract(Me.stampedTime)

        'convert Now time to whole seconds
        Dim nowtimesec As Long = CLng(nowTime.TotalSeconds)  'truncate the fractional seconds
        nowTime = New TimeSpan(nowtimesec * 10000000)


        'call the test unit at CERL once an hour at half past the hour
        'If nowTime = New TimeSpan(0, 0, 0) Then 'subtract 24 hours - lets test run after midnight
        '    stampedTime2 = stampedTime2.Subtract(New TimeSpan(24, 0, 0))
        'End If

        'Dim diffTime2 As TimeSpan = nowTime.Subtract(Me.stampedTime2)
```

```vb
        'If diffTime2 > New TimeSpan(1, 0, 0) Then
        '    'call unit 99 at 15 mins past the hour, every hour

        '    SyncLock LDUnitList
        '        For Each u As LDUnit In LDUnitList
        '            If u.UnitNum = "L99" OrElse u.UnitNum = "L05" Then
        '                u.IsDownloadDone = False
        '                u.IsDownloadInProgress = False
        '            Else
        '                u.IsDownloadDone = True
        '            End If
        '        Next
        '    End SyncLock

        '    CheckBox1.Checked = True
        '    CheckBox2.Checked = True
        '    CheckBox3.Checked = True
        '    CheckBox4.Checked = False

        '    If Not Me.BWDelayedClick.IsBusy Then
        '        Me.BWDelayedClick.RunWorkerAsync()    'dial out
        '    Else
        '        Dim eaM As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingText)
        '        eaM.PrecedingText = "Cannot start download at " & Date.Now.ToString("M/d/yyyy h:mm:ss tt") _
        '        & "due to a busy worker thread." & vbLf & vbLf
        '        RaiseEvent UpdateMessages(Me, eaM)
        '    End If

        '    Me.stampedTime2 = nowTime

        'End If
        'end test

        Select Case nowDay
            Case DayOfWeek.Sunday
                If appset.IsSunAWorkday Then
                    If Form1.appset.WorkdayStart <= Form1.appset.WorkdayEnd Then
                        If nowTime > Form1.appset.WorkdayStart AndAlso nowTime < Form1.appset.WorkdayEnd Then
                            IsDaytime = True
                        Else
                            IsDaytime = False
                        End If
                    Else 'ex. day starts at 6am and ends at 1am
                        If nowTime > Form1.appset.WorkdayEnd AndAlso nowTime < Form1.appset.WorkdayStart Then
                            IsDaytime = False
                        Else
                            IsDaytime = True
                        End If
                    End If
                    WorkdaySched(nowTime, diffTime)
                Else
                    IsDaytime = False
                    WeekendSched(nowTime)
                End If
            Case DayOfWeek.Saturday
                If appset.IsSatAWorkday Then
                    If Form1.appset.WorkdayStart <= Form1.appset.WorkdayEnd Then
                        If nowTime > Form1.appset.WorkdayStart AndAlso nowTime < Form1.appset.WorkdayEnd Then
                            IsDaytime = True
                        Else
                            IsDaytime = False
                        End If
                    Else
                        If nowTime > Form1.appset.WorkdayEnd AndAlso nowTime < Form1.appset.WorkdayStart Then
                            IsDaytime = False
                        Else
                            IsDaytime = True
                        End If
                    End If
                    WorkdaySched(nowTime, diffTime)
                Else
                    IsDaytime = False
                    WeekendSched(nowTime)
                End If
                'otherwise
            Case Else
                If Form1.appset.WorkdayStart <= Form1.appset.WorkdayEnd Then
                    If nowTime > Form1.appset.WorkdayStart AndAlso nowTime < Form1.appset.WorkdayEnd Then
                        IsDaytime = True
                    Else
                        IsDaytime = False
                    End If
                Else 'ex. day starts at 6am and ends at 1am
                    If nowTime > Form1.appset.WorkdayEnd AndAlso nowTime < Form1.appset.WorkdayStart Then
                        IsDaytime = False
                    Else
                        IsDaytime = True
                    End If
                End If
                WorkdaySched(nowTime, diffTime)

        End Select

    End Sub

    Private Sub WorkdaySched(ByVal nowtime As TimeSpan, ByVal difftime As TimeSpan)
        Dim e As New EventArgs

        Select Case nowtime
            Case appset.WeeknightDL '10:30pm by default
                'nightly download for all units - they should be stopped by timers by now
                Dim uList As List(Of LDUnit) = GetAllUnits()
                For Each unit As LDUnit In uList
                    If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                        unit.AllowCallIns = True
                        unit.ResetDataYN = True
                        unit.ResetTimeYN = True 'uncomment when APG units are put on timer
                        unit.DLTries = 0
                        unit.IncludeLC = False
                        unit.IncludeQ = False
                        unit.IncludeR = False
                        unit.IsDownloadDone = False
                        unit.IsDownloadInProgress = False
                    Else
                        unit.AllowCallIns = False
                        unit.ResetDataYN = True
                        unit.ResetTimeYN = True
                        unit.DLTries = 0
                        unit.IncludeLC = False
                        unit.IncludeQ = False
                        unit.IncludeR = False
                        unit.IsDownloadDone = False
                        unit.IsDownloadInProgress = False
                    End If
                    'desired exceedance threshold is now determined in the Modem.Download method
                Next
                ReturnAllUnits(uList)
                CheckBox1.Checked = False
                CheckBox2.Checked = False
                CheckBox3.Checked = False  'no units should be calling in now
                CheckBox4.Checked = False

                If Not Me.BWDelayedClick.IsBusy Then
                    Me.BWDelayedClick.RunWorkerAsync()    'dial out
                Else
                    Dim eaM As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingText)
                    eaM.PrecedingText = "Cannot start download at " & Date.Now.ToString("M/d/yyyy h:mm:ss tt↙
    ") _
                    & "due to a busy worker thread." & vbLf & vbLf
                    RaiseEvent UpdateMessages(Me, eaM)
                End If




                'Case appset.WorkdayStart '6am by default, soon to be obsolete
                '    'call APG units, enable dial-out mode
                '    'lower exceedance threshold to 100 dB for all units
                '    'Form1.IsDaytime = True

                '    Dim uList As List(Of LDUnit) = GetAllUnits()
                '    For Each unit As LDUnit In uList
                '        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                '            unit.AllowCallIns = True
                '            unit.ResetDataYN = True
                '            unit.IncludeQ = False
                '            unit.IncludeR = False
                '            unit.IsDownloadDone = False
                '            unit.IsDownloadInProgress = False
                '            'unit.ExcdThreshold = appset.WorkdayExcdThresh
                '        Else 'don't call
                '            unit.IsDownloadDone = True
                '            unit.AllowCallIns = False
                '            unit.ResetDataYN = True
                '            unit.IncludeQ = False
                '            unit.IncludeR = False
                '            'unit.ExcdThreshold = appset.WorkdayExcdThresh
                '        End If
                '    Next
                '    ReturnAllUnits(uList)

                '    'initialize the Listen timer
                '    stampedTime = nowtime

                '    'save the text from the night download 'is now done after every download
                '    'SaveFileDialog1.FileName = appset.InstallPath & "Logs\Log " & _
                '    'Date.Now.ToString("ddMMMyyyy HHmm") & ".rtf"

                '    'Dim ea2 As New CancelEventArgs(False)
                '    'SaveFileDialog1_FileOk(Me, ea2)

                '    CheckBox1.Checked = False
                '    CheckBox2.Checked = False
                '    CheckBox3.Checked = True    'leave line open to receive calls from units who could dial↙
     in
                '    CheckBox4.Checked = False

                '    If Not Me.BWDelayedClick.IsBusy Then
                '        Me.BWDelayedClick.RunWorkerAsync()    'dial out
                '    Else
                '        Dim eaM As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingText)
                '        eaM.PrecedingText = "Cannot start download at " & Date.Now.ToString("M/d/yyyy h:mm:↙
    ss tt") _
                '        & "due to a busy worker thread." & vbLf & vbLf
                '        RaiseEvent UpdateMessages(Me, eaM)
                '    End If


                'Case appset.WorkdayEnd '10pm by default, replace with Workday Night Download at 10pm
                '    '(so comment out this section and WorkdayStart)

                '    'call APG units, disable dial-out mode
                '    'Form1.IsDaytime = False

                '    Dim uList As List(Of LDUnit) = GetAllUnits()
                '    For Each unit As LDUnit In uList
                '        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                '            unit.AllowCallIns = False 'change to True when APG units are put on the timer
                '            unit.ResetDataYN = True
                '            'unit.ResetTimeYN = True 'uncomment when APG units are put on timer
                '            unit.DLTries = 0
                '            unit.IncludeLC = False
                '            unit.IncludeQ = False
                '            unit.IncludeR = False
                '            unit.IsDownloadDone = False
                '            unit.IsDownloadInProgress = False
                '            'unit.ExcdThreshold = appset.OtherExcdThresh
                '        Else
                '            unit.AllowCallIns = False
                '            unit.ResetDataYN = True
                '            unit.ResetTimeYN = True
                '            unit.DLTries = 0
                '            unit.IncludeLC = False
                '            unit.IncludeQ = False
                '            unit.IncludeR = False
                '            unit.IsDownloadDone = False
                '            unit.IsDownloadInProgress = False
                '            'unit.ExcdThreshold = appset.OtherExcdThresh
                '        End If
                '    Next
                '    ReturnAllUnits(uList)
                '    CheckBox1.Checked = False
                '    CheckBox2.Checked = False
                '    CheckBox3.Checked = True    'this line is the first to receive calls
                '    CheckBox4.Checked = False

                '    If Not Me.BWDelayedClick.IsBusy Then
                '        Me.BWDelayedClick.RunWorkerAsync()    'dial out
                '    Else
                '        Dim eaM As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingText)
                '        eaM.PrecedingText = "Cannot start download at " & Date.Now.ToString("M/d/yyyy h:mm:↙
    ss tt") _
                '        & "due to a busy worker thread." & vbLf & vbLf
                '        RaiseEvent UpdateMessages(Me, eaM)
                '    End If


        End Select

        're-check the Listen boxes
        If appset.DialInProg Then
            Me.stampedTime = nowtime
        End If

        If (nowtime > appset.WorkdayStart.Add(New TimeSpan(0, 30, 0)) AndAlso nowtime < appset.WorkdayEnd _
        AndAlso (difftime > New TimeSpan(0, 25, 0)) AndAlso Not appset.DialInProg) Then
            If CheckBox1.Enabled Then
                CheckBox1.Checked = True
            End If
            If CheckBox2.Enabled Then
                CheckBox2.Checked = True
            End If
            If CheckBox3.Enabled Then
                CheckBox3.Checked = True
            End If
            If CheckBox4.Enabled Then
                CheckBox4.Checked = True
            End If
            Me.stampedTime = nowtime
        End If

    End Sub

    Private Sub WeekendSched(ByVal nowtime As TimeSpan)
        'Dim nightlyDL1 As New TimeSpan(21, 0, 0)    'use for test only
        Dim e As New EventArgs

        Select Case nowtime
            Case appset.WeekendDL 'change this to 10pm
                Dim uList As List(Of LDUnit) = GetAllUnits()
                For Each unit As LDUnit In uList
                    unit.ResetDataYN = True
                    unit.ResetTimeYN = True
                    unit.DLTries = 0
                    unit.IncludeLC = False
                    unit.IncludeQ = True
                    unit.IncludeR = True
                    unit.IsDownloadDone = False
                    unit.IsDownloadInProgress = False
                Next
                ReturnAllUnits(uList)

                If Not Me.BWDelayedClick.IsBusy Then
                    Me.BWDelayedClick.RunWorkerAsync()    'dial out
                Else
                    Dim eaM As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ChangePrecedingText)
                    eaM.PrecedingText = "Cannot start download at " & Date.Now.ToString("M/d/yyyy h:mm:ss tt↙
    ") _
                    & "due to a busy worker thread." & vbLf & vbLf
                    RaiseEvent UpdateMessages(Me, eaM)
                End If

        End Select
    End Sub

#End Region

#Region "Get/Return/Save Units"
    Public Shared Function GetUnit(ByVal LNum As String) As LDUnit
        'tries to return the unit specified by LNum
        'if LNum = -1, returns the next available unit
        'If Dial wants to get the next available unit, it needs to pass -2 as the unit number
        '   (this is an obsolete requirement - -1 and -2 do the same thing now)
        'Conversely, Listen will only be able to grab a specific unit if call-ins are enabled on that unit

        Dim unitindex As Integer = 0
        Dim unit As LDUnit
        Dim unitToReturn As LDUnit 'use to ensure End Synclock is always called

        SyncLock LDUnitList
            If String.Compare(LNum, "-1") = 0 OrElse String.Compare(LNum, "-2") = 0 Then
                unitindex = LDUnitList.FindIndex(AddressOf NextUnit)
                If unitindex > -1 Then
                    unit = LDUnitList.Item(unitindex)
                    LDUnitList.RemoveAt(unitindex)
                    unitToReturn = unit
                Else    'no available units
                    unitToReturn = New LDUnit("null")
                End If
            Else
                currLNum = LNum
                unitindex = LDUnitList.FindIndex(AddressOf SpecificUnit)
                If unitindex > -1 Then
                    unit = LDUnitList.Item(unitindex)
                    LDUnitList.RemoveAt(unitindex)
                    unitToReturn = unit
                Else
                    unitToReturn = New LDUnit("null")
                End If
            End If
        End SyncLock

        If Form1.IsUVOpen Then
            uv.RefreshToolStripMenuItem.PerformClick()
        End If

        Return unitToReturn

    End Function

    Public Shared Sub ReturnUnit(ByVal unit As LDUnit)
        SyncLock LDUnitList
            unit.DLTries = 0
            LDUnitList.Add(unit)
        End SyncLock

        If Form1.IsUVOpen Then
            uv.RefreshToolStripMenuItem.PerformClick()
        End If
    End Sub

    Private Shared Function NextUnit(ByVal unit As LDUnit) As Boolean
        If unit.IsDownloadDone = False AndAlso unit.IsDownloadInProgress = False Then
```

```
            Return True
        Else
            Return False
        End If
    End Function

    Private Shared Function SpecificUnit(ByVal unit As LDUnit) As Boolean
        If String.Compare(unit.UnitNum, currLNum) = 0 Then
            Return True
        Else
            Return False
        End If
    End Function

    Private Shared Function NullUnit(ByVal unit As LDUnit) As Boolean
        If unit.UnitNum = "null" Then
            Return True
        Else
            Return False
        End If
    End Function

    'Public Shared Function GetUnits() As List(Of LDUnit)
    '    'Gets only the units marked for download - used by Button2

    '    Dim uList As New List(Of LDUnit)
    '    SyncLock LDUnitList
    '        Do
    '            uList.Add(GetUnit("-1"))
    '        Loop Until uList.Exists(AddressOf NullUnit)
    '    End SyncLock
    '    'get rid of the returned null unit
    '    uList.RemoveAll(AddressOf NullUnit)

    '    Return uList

    'End Function

    Public Shared Function GetAllUnits() As List(Of LDUnit)
        Dim uList As New List(Of LDUnit)

        MakeUnitsDownloadable()
        SyncLock LDUnitList
            uList.AddRange(LDUnitList)
            LDUnitList.Clear()
        End SyncLock
        'get rid of the returned null unit
        uList.RemoveAll(AddressOf NullUnit)

        Return uList

    End Function

    Public Shared Function GetAllUnitsNoMark() As List(Of LDUnit)
        'gets all the units without changing their markings
        Dim ulist As New List(Of LDUnit)

        SyncLock LDUnitList
            ulist.AddRange(LDUnitList)
            LDUnitList.Clear()
        End SyncLock

        Return ulist
    End Function

    Public Shared Sub ReturnAllUnits(ByRef uList As List(Of LDUnit))
        For Each unit As LDUnit In uList
            ReturnUnit(unit)
        Next
        'let garbage collection handle uList
    End Sub

    Private Shared Sub MakeUnitsDownloadable()
        'run on Form1 thread?

        SyncLock LDUnitList
            Dim remAtIndex As New List(Of Integer)
            For i As Integer = 0 To LDUnitList.Count - 1
                LDUnitList.Item(i).IsDownloadDone = False
                LDUnitList.Item(i).IsDownloadInProgress = False
                LDUnitList.Item(i).DialTries = 0
                If LDUnitList.Item(i).UnitNum = "null" OrElse LDUnitList.Item(i).UnitNum = "temp" Then
                    remAtIndex.Add(i)
                End If
            Next

            Dim res As String = ""
            For Each inte As Integer In remAtIndex
                res &= inte.ToString & "  "
            Next
            'MessageBox.Show(res)
            For j As Integer = 0 To remAtIndex.Count - 1
```

```
                LDUnitList.RemoveAt(remAtIndex.Item(j))
            Next
        End SyncLock
    End Sub

    Private Sub SaveLDUnitList()
        Dim ucomp As New uCompare()
        Dim comp As Collections.Generic.IComparer(Of LDUnit) = ucomp
        SyncLock LDUnitList
            LDUnitList.Sort(comp)
            LDUnitList.TrimExcess()
            For Each unit As LDUnit In LDUnitList
                With unit
                    .EList.Clear()
                    .IList.Clear()
                    .RList.Clear()
                    .QList.Clear()
                    .LList.Clear()
                    .CList.Clear()
                End With
            Next

            Using fs As New FileStream(appset.InstallPath & "units.dat", FileMode.Create)

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                bf.Serialize(fs, LDUnitList)
            End Using
        End SyncLock
    End Sub

#End Region

#Region "Dial/Listen Control"

    Private Sub Button2_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button2.TextChanged
        'hide button if text = Dial - makes interface less confusing
        If Button2.Text = "Dial" Then
            Button2.Visible = False
            UnitOptionsToolStripMenuItem.Enabled = True
            Me.UnitOptionsToolStripMenuItem1.Enabled = True
        Else
            Button2.Visible = True
            UnitOptionsToolStripMenuItem.Enabled = False
            Me.UnitOptionsToolStripMenuItem1.Enabled = False
        End If
    End Sub

    Public Delegate Sub PerformClickDelegate()
    Private Sub performclick()
        Dim e As New EventArgs()
        Button2_Click(Me, e)
    End Sub
    Private Sub BWDelayedClick_DoWork(ByVal sender As Object, ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BWDelayedClick.DoWork
        'performs a delayed button2 click without blocking the UI thread
        Dim pcdeleg As New PerformClickDelegate(AddressOf performclick)
        Dim normalDeleg As New ChangeCursor(AddressOf normalCursor)
        Dim waitDeleg As New ChangeCursor(AddressOf waitCursor)

        Me.Invoke(waitDeleg)
        Thread.Sleep(5000)
        Me.Invoke(normalDeleg)

        If Button2.Text <> "Dial" Then
            Button2.Invoke(pcdeleg)
            Me.Invoke(waitDeleg)
            Thread.Sleep(5000)
            Me.Invoke(normalDeleg)
            Button2.Invoke(pcdeleg)
        Else
            Button2.Invoke(pcdeleg)
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        'Dial units using available modems
        'Insert code to determine which LDUnits to dial, and not to dial modems which are in Listen mode
        If Button2.Text = "Dial" Then
            appset.DialInProg = True 'a dial operation is in progress

            If Not sender.Equals(Me) Then
                MakeUnitsDownloadable()
            End If

            'tell status bar how many units are left
            Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Populate)
            RaiseEvent SBUnitCounter(Me, EA)
            RaiseEvent UpdateMessages(Me, EA)

            'run the background workers
            If modem1.modemMode = Modem.Mode.Standby AndAlso Not BW1.IsBusy Then
                txtOut1.Clear()
                txtOut1h.Clear()
                rtfBody1.Clear()
                modem1.modemMode = Modem.Mode.Dial
                BW1.RunWorkerAsync()
            End If
            If modem2.modemMode = Modem.Mode.Standby AndAlso Not BW2.IsBusy Then
                txtOut2.Clear()
                txtOut2h.Clear()
                rtfBody2.Clear()
                modem2.modemMode = Modem.Mode.Dial
                BW2.RunWorkerAsync()
            End If
            If modem3.modemMode = Modem.Mode.Standby AndAlso Not BW3.IsBusy Then
                txtOut3.Clear()
                txtOut3h.Clear()
                rtfBody3.Clear()
                modem3.modemMode = Modem.Mode.Dial
                BW3.RunWorkerAsync()
            End If
            If modem4.modemMode = Modem.Mode.Standby AndAlso Not BW4.IsBusy Then
                txtOut4.Clear()
                txtOut4h.Clear()
                rtfBody4.Clear()
                modem4.modemMode = Modem.Mode.Dial
                BW4.RunWorkerAsync()
            End If
            Button2.Text = "Cancel Downloads"

        ElseIf Button2.Text <> "Dial" Then
            appset.DialInProg = False
            Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Clear)
            RaiseEvent SBUnitCounter(Me, EA)

            If modem1.modemMode = Modem.Mode.Dial Then
                BW1.CancelAsync()
            End If
            If modem2.modemMode = Modem.Mode.Dial Then
                BW2.CancelAsync()
            End If
            If modem3.modemMode = Modem.Mode.Dial Then
                BW3.CancelAsync()
            End If
            If modem4.modemMode = Modem.Mode.Dial Then
                BW4.CancelAsync()
            End If
            If modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen _
            AndAlso modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen Then
                Button2.Text = "Dial"
            End If
            Button2.Enabled = False
        End If


    End Sub

    Private Sub BW1_DoWork(ByVal sender As System.Object, ByVal e As System.ComponentModel.DoWorkEventArgs) ↙
    Handles BW1.DoWork
        'this method does not run on the UI thread
        Dim worker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

        If modem1.modemMode = Modem.Mode.Dial Then
            modem1.Dial(worker, e)
        ElseIf modem1.modemMode = Modem.Mode.Listen Then
            modem1.Listen(worker, e)
        End If

    End Sub
    Private Sub BW1_RunWorkerCompleted(ByVal sender As System.Object, ByVal e As System.ComponentModel.    ↙
    RunWorkerCompletedEventArgs) Handles BW1.RunWorkerCompleted
        'this method runs on the UI thread
        'This can either be reached via a normal completion or a user cancel

        If e.Cancelled Then 'process was canceled either by a Listen check or a Cancel Download click
            CheckBox1.Enabled = True
            If modem1.modemMode = Modem.Mode.Listen Then
                BW1.RunWorkerAsync()
            ElseIf modem1.modemMode = Modem.Mode.ListenCancel Then
                'the listen box for this modem was just unchecked
                If appset.DialInProg Then
                    'a dial is in progress
                    modem1.modemMode = Modem.Mode.Dial
                    BW1.RunWorkerAsync()
                Else
                    modem1.modemMode = Modem.Mode.Standby
                    'do nothing else
                End If
            Else
                'a dial cancel was ordered by the user
                modem1.modemMode = Modem.Mode.Standby
                If Not (modem2.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
                OrElse modem4.modemMode = Modem.Mode.Dial) Then
                    Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Cancelled)
                    RaiseEvent UpdateMessages(Me, EA)
                End If
            End If

            If Not (modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen _
            AndAlso modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen) Then
                'it is appropriate to re-enable the dial button and menu
                Button2.Enabled = True
                Me.DownloadOneUnitToolStripMenuItem.Enabled = True
            End If

        Else
            'the process exited normally (if modem is in Dial mode)
            If modem1.modemMode = Modem.Mode.Dial Then
                modem1.modemMode = Modem.Mode.Standby
            End If

            If modem1.modemMode = Modem.Mode.Standby AndAlso Not appset.DialInProg AndAlso Not _
                (modem2.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
                OrElse modem4.modemMode = Modem.Mode.Dial) Then
                Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ShowText)
                RaiseEvent UpdateMessages(Me, EA)
                EA.Action = StatusBarEventArgs.SBAction.HideText
                RaiseEvent SBUnitCounter(Me, EA)

                'log the info on the screen to a file
                If appset.SaveLogAfterDL Then
                    SaveFileDialog1.FileName = appset.InstallPath & "Logs\Log " & _
                        Date.Now.ToString("ddMMMyyyy HHmm") & ".rtf"
                    Dim ea2 As New CancelEventArgs(False)
                    SaveFileDialog1_FileOk(Me, ea2)
                End If

                'enable the Listen boxes during the day
                ReEnableListen()
            End If

        End If

        'if all modems are out of Dial mode, change button2 text to Dial and re-enable the dial button and ↙
    menu
        If Not (modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
            OrElse modem3.modemMode = Modem.Mode.Dial OrElse modem4.modemMode = Modem.Mode.Dial) Then
            'appset.DialInProg = False
            Button2.Text = "Dial"
            Button2.Enabled = True
            Me.DownloadOneUnitToolStripMenuItem.Enabled = True
        End If

    End Sub
    Private Sub BW2_DoWork(ByVal sender As System.Object, ByVal e As System.ComponentModel.DoWorkEventArgs) ↙
    Handles BW2.DoWork

        Dim worker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

        If modem2.modemMode = Modem.Mode.Dial Then
            modem2.Dial(worker, e)
        ElseIf modem2.modemMode = Modem.Mode.Listen Then
            modem2.Listen(worker, e)
        End If

    End Sub
    Private Sub BW2_RunWorkerCompleted(ByVal sender As System.Object, ByVal e As System.ComponentModel.      ↙
    RunWorkerCompletedEventArgs) Handles BW2.RunWorkerCompleted
        'this method runs on the UI thread
        'This can either be reached via a normal completion, an exception, or a user cancel
        If e.Cancelled Then 'process was canceled either by a Listen check or a Cancel Download click
            CheckBox2.Enabled = True
            If modem2.modemMode = Modem.Mode.Listen Then
                BW2.RunWorkerAsync()
            ElseIf modem2.modemMode = Modem.Mode.ListenCancel Then
                'the listen box for this modem was just unchecked
                If appset.DialInProg Then
                    'a dial is in progress
                    modem2.modemMode = Modem.Mode.Dial
                    BW2.RunWorkerAsync()
                Else
                    modem2.modemMode = Modem.Mode.Standby
                    'do nothing else
                End If
            Else
                'a dial cancel was ordered by the user
                modem2.modemMode = Modem.Mode.Standby
                If Not (modem1.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
                OrElse modem4.modemMode = Modem.Mode.Dial) Then
                    Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Cancelled)
                    RaiseEvent UpdateMessages(Me, EA)
                End If
            End If

            If Not (modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen _
            AndAlso modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen) Then
                'it is appropriate to re-enable the dial button and menu
                Button2.Enabled = True
                Me.DownloadOneUnitToolStripMenuItem.Enabled = True
            End If

        Else
            'the process exited normally (if modem is in Dial mode)
            If modem2.modemMode = Modem.Mode.Dial Then
                modem2.modemMode = Modem.Mode.Standby
            End If

            If modem2.modemMode = Modem.Mode.Standby AndAlso Not appset.DialInProg AndAlso Not _
            (modem1.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
            OrElse modem4.modemMode = Modem.Mode.Dial) Then 'all modems finished dialing
                Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ShowText)
                RaiseEvent UpdateMessages(Me, EA)
                EA.Action = StatusBarEventArgs.SBAction.HideText
                RaiseEvent SBUnitCounter(Me, EA)

                'log the info on the screen to a file
                If appset.SaveLogAfterDL Then
                    SaveFileDialog1.FileName = appset.InstallPath & "Logs\Log " & _
                        Date.Now.ToString("ddMMMyyyy HHmm") & ".rtf"
                    Dim ea2 As New CancelEventArgs(False)
                    SaveFileDialog1_FileOk(Me, ea2)
                End If

                'enable the Listen boxes during the day
                ReEnableListen()
            End If
        End If

        'if all modems are out of Dial mode, change button2 text to Dial and re-enable the dial button and ↙
    menu
        If Not (modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
            OrElse modem3.modemMode = Modem.Mode.Dial OrElse modem4.modemMode = Modem.Mode.Dial) Then
            appset.DialInProg = False
            Button2.Text = "Dial"
            Button2.Enabled = True
            Me.DownloadOneUnitToolStripMenuItem.Enabled = True
        End If

    End Sub
    Private Sub BW3_DoWork(ByVal sender As System.Object, ByVal e As System.ComponentModel.DoWorkEventArgs) ↙
    Handles BW3.DoWork

        Dim worker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

        If modem3.modemMode = Modem.Mode.Dial Then
            modem3.Dial(worker, e)
        ElseIf modem3.modemMode = Modem.Mode.Listen Then
            modem3.Listen(worker, e)
        End If

    End Sub
    Private Sub BW3_RunWorkerCompleted(ByVal sender As System.Object, ByVal e As System.ComponentModel. ↙
    RunWorkerCompletedEventArgs) Handles BW3.RunWorkerCompleted
        'this method runs on the UI thread
        'This can either be reached via a normal completion, an exception, or a user cancel

        If e.Cancelled Then 'process was canceled either by a Listen check or a Cancel Download click
            CheckBox3.Enabled = True
            If modem3.modemMode = Modem.Mode.Listen Then
                BW3.RunWorkerAsync()
            ElseIf modem3.modemMode = Modem.Mode.ListenCancel Then
                'the listen box for this modem was just unchecked
                If appset.DialInProg Then
                    'a dial is in progress
                    modem3.modemMode = Modem.Mode.Dial
                    BW3.RunWorkerAsync()
                Else
                    modem3.modemMode = Modem.Mode.Standby
                    'do nothing else
                End If
            Else
                'a dial cancel was ordered by the user
                modem3.modemMode = Modem.Mode.Standby
                If Not (modem2.modemMode = Modem.Mode.Dial OrElse modem1.modemMode = Modem.Mode.Dial _
                OrElse modem4.modemMode = Modem.Mode.Dial) Then
                    Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Cancelled)
                    RaiseEvent UpdateMessages(Me, EA)
                End If
            End If

            If Not (modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen _
            AndAlso modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen) Then
                'it is appropriate to re-enable the dial button and menu
                Button2.Enabled = True
                Me.DownloadOneUnitToolStripMenuItem.Enabled = True
            End If
```

```vb

        Else
            'the process exited normally (if modem is in Dial mode)
            If modem3.modemMode = Modem.Mode.Dial Then
                modem3.modemMode = Modem.Mode.Standby
            End If

            If modem3.modemMode = Modem.Mode.Standby AndAlso Not appset.DialInProg AndAlso Not _
            (modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
            OrElse modem4.modemMode = Modem.Mode.Dial) Then 'dial sequence is done
                Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ShowText)
                RaiseEvent UpdateMessages(Me, EA)
                EA.Action = StatusBarEventArgs.SBAction.HideText
                RaiseEvent SBUnitCounter(Me, EA)

                'log the info on the screen to a file
                If appset.SaveLogAfterDL Then
                    SaveFileDialog1.FileName = appset.InstallPath & "Logs\Log " & _
                        Date.Now.ToString("ddMMMyyyy HHmm") & ".rtf"
                    Dim ea2 As New CancelEventArgs(False)
                    SaveFileDialog1_FileOk(Me, ea2)
                End If

                'enable the Listen boxes during the day
                ReEnableListen()
            End If
        End If

        'if all modems are out of Dial mode, change button2 text to Dial and re-enable the dial button and ↙
    menu
        If Not (modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
            OrElse modem3.modemMode = Modem.Mode.Dial OrElse modem4.modemMode = Modem.Mode.Dial) Then
            appset.DialInProg = False
            Button2.Text = "Dial"
            Button2.Enabled = True
            Me.DownloadOneUnitToolStripMenuItem.Enabled = True
        End If
    End Sub
    Private Sub BW4_DoWork(ByVal sender As System.Object, ByVal e As System.ComponentModel.DoWorkEventArgs) ↙
    Handles BW4.DoWork

        Dim worker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

        If modem4.modemMode = Modem.Mode.Dial Then
            modem4.Dial(worker, e)
        ElseIf modem4.modemMode = Modem.Mode.Listen Then
            modem4.Listen(worker, e)
        End If

    End Sub
    Private Sub BW4_RunWorkerCompleted(ByVal sender As System.Object, ByVal e As System.ComponentModel.    ↙
    RunWorkerCompletedEventArgs) Handles BW4.RunWorkerCompleted
        'this method runs on the UI thread
        'This can either be reached via a normal completion, an exception, or a user cancel

        If e.Cancelled Then 'process was canceled either by a Listen check or a Cancel Download click
            CheckBox4.Enabled = True
            If modem4.modemMode = Modem.Mode.Listen Then
                BW4.RunWorkerAsync()
            ElseIf modem4.modemMode = Modem.Mode.ListenCancel Then
                'the listen box for this modem was just unchecked
                If appset.DialInProg Then
                    'a dial is in progress
                    modem4.modemMode = Modem.Mode.Dial
                    BW4.RunWorkerAsync()
                Else
                    modem4.modemMode = Modem.Mode.Standby
                    'do nothing else
                End If
            Else
                'a dial or listen cancel was ordered by the user
                modem4.modemMode = Modem.Mode.Standby
                If Not (modem2.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
                OrElse modem1.modemMode = Modem.Mode.Dial) Then
                    Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Cancelled)
                    RaiseEvent UpdateMessages(Me, EA)
                End If
            End If

            If Not (modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen _
            AndAlso modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen) Then
                'it is appropriate to re-enable the dial button and menu
                Button2.Enabled = True
                Me.DownloadOneUnitToolStripMenuItem.Enabled = True
            End If

        Else
            'the process exited normally (if modem is in Dial mode)
            If modem4.modemMode = Modem.Mode.Dial Then
                modem4.modemMode = Modem.Mode.Standby
            End If

            If modem4.modemMode = Modem.Mode.Standby AndAlso Not appset.DialInProg AndAlso Not _
            (modem2.modemMode = Modem.Mode.Dial OrElse modem3.modemMode = Modem.Mode.Dial _
            OrElse modem1.modemMode = Modem.Mode.Dial) Then 'dial process has completed
                Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.ShowText)
                RaiseEvent UpdateMessages(Me, EA)
                EA.Action = StatusBarEventArgs.SBAction.HideText
                RaiseEvent SBUnitCounter(Me, EA)

                'log the info on the screen to a file
                If appset.SaveLogAfterDL Then
                    SaveFileDialog1.FileName = appset.InstallPath & "Logs\Log " & _
                        Date.Now.ToString("ddMMMyyyy HHmm") & ".rtf"
                    Dim ea2 As New CancelEventArgs(False)
                    SaveFileDialog1_FileOk(Me, ea2)
                End If

                'enable the Listen boxes during the day
                ReEnableListen()
            End If
        End If

        'if all modems are out of Dial mode, change button2 text to Dial and re-enable the dial button and 
    menu
        If Not (modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
            OrElse modem3.modemMode = Modem.Mode.Dial OrElse modem4.modemMode = Modem.Mode.Dial) Then
            appset.DialInProg = False
            Button2.Text = "Dial"
            Button2.Enabled = True
            Me.DownloadOneUnitToolStripMenuItem.Enabled = True
        End If

    End Sub

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
     CheckBox1.CheckedChanged

        If CheckBox1.Checked Then
            modem1.modemMode = Modem.Mode.Listen
            If BW1.IsBusy = False Then
                BW1.RunWorkerAsync()
            Else
                'attempt to cancel the dial operation, then Listen
                CheckBox1.Enabled = False
                BW1.CancelAsync()
            End If
            If modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen AndAlso _
            modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen Then
                Button2.Enabled = False
            End If
        Else
            'listen was deactivated, BW2 should always be busy
            If BW1.IsBusy Then
                CheckBox1.Enabled = False
                modem1.modemMode = Modem.Mode.ListenCancel
                BW1.CancelAsync()
            End If

        End If
    End Sub

    Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
     CheckBox2.CheckedChanged
        If CheckBox2.Checked Then
            modem2.modemMode = Modem.Mode.Listen
            If BW2.IsBusy = False Then
                BW2.RunWorkerAsync()
            Else
                'attempt to cancel the dial operation, then Listen
                CheckBox2.Enabled = False
                BW2.CancelAsync()
            End If
            If modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen AndAlso _
            modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen Then
                Button2.Enabled = False
            End If
        Else
            'listen was deactivated, BW2 should always be busy
            If BW2.IsBusy Then
                CheckBox2.Enabled = False
                modem2.modemMode = Modem.Mode.ListenCancel
                BW2.CancelAsync()
            End If

        End If

    End Sub

    Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
     CheckBox3.CheckedChanged

        If CheckBox3.Checked Then
            modem3.modemMode = Modem.Mode.Listen
```

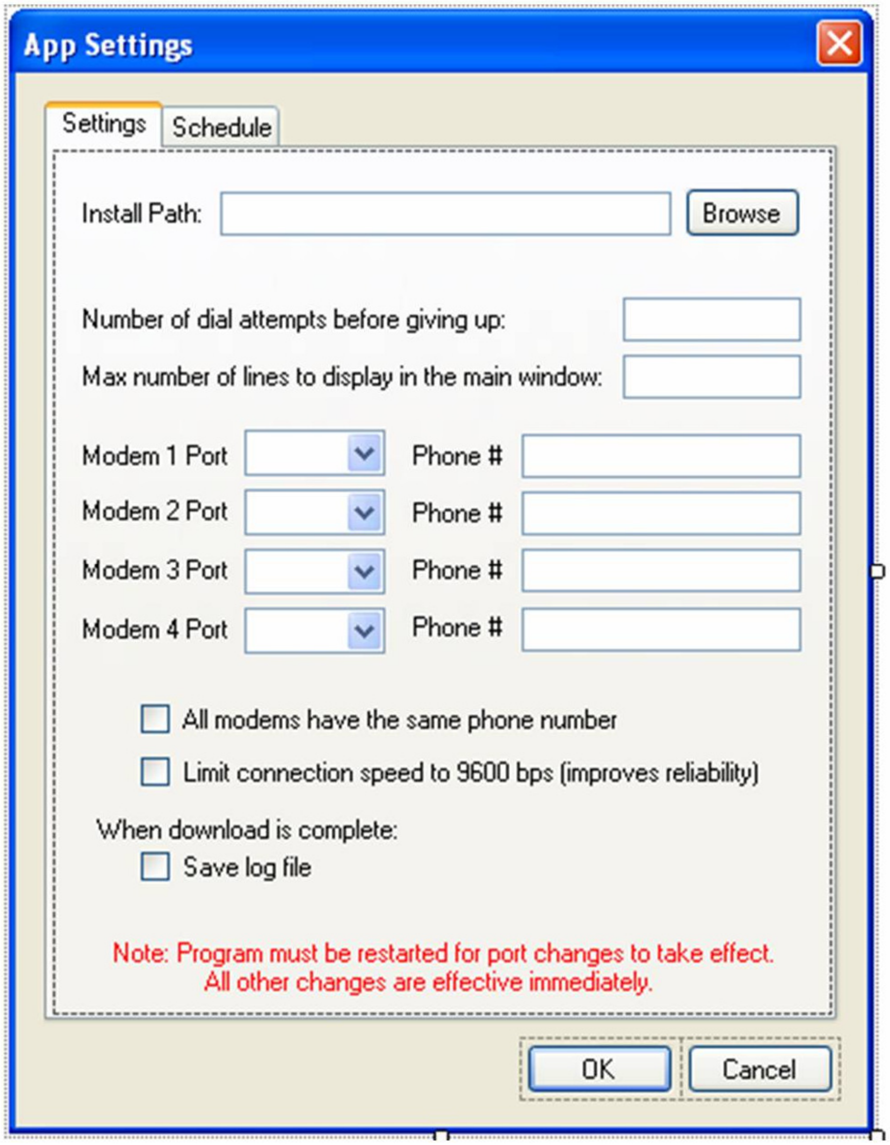
```vb
            If BW3.IsBusy = False Then
                BW3.RunWorkerAsync()
            Else
                'attempt to cancel the dial operation, then Listen
                CheckBox3.Enabled = False
                BW3.CancelAsync()
            End If
            If modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen AndAlso _
            modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen Then
                Button2.Enabled = False
            End If
        Else
            'listen was deactivated, BW2 should always be busy
            If BW3.IsBusy Then
                CheckBox3.Enabled = False
                modem3.modemMode = Modem.Mode.ListenCancel
                BW3.CancelAsync()
            End If

        End If
    End Sub

    Private Sub CheckBox4_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox4.CheckedChanged


        If CheckBox4.Checked Then
            modem4.modemMode = Modem.Mode.Listen
            If BW4.IsBusy = False Then
                BW4.RunWorkerAsync()
            Else
                'attempt to cancel the dial operation, then Listen
                CheckBox4.Enabled = False
                BW4.CancelAsync()
            End If
            If modem1.modemMode = Modem.Mode.Listen AndAlso modem2.modemMode = Modem.Mode.Listen AndAlso _
            modem3.modemMode = Modem.Mode.Listen AndAlso modem4.modemMode = Modem.Mode.Listen Then
                Button2.Enabled = False
            End If
        Else
            'listen was deactivated, BW2 should always be busy
            If BW4.IsBusy Then
                CheckBox4.Enabled = False
                modem4.modemMode = Modem.Mode.ListenCancel
                BW4.CancelAsync()
            End If

        End If
    End Sub


    Private Sub CheckBox5_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox5.CheckedChanged
        If CheckBox5.Checked Then
            modem1.speakerOn = True
        Else
            modem1.speakerOn = False
        End If
    End Sub

    Private Sub CheckBox6_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox6.CheckedChanged
        If CheckBox6.Checked Then
            modem2.speakerOn = True
        Else
            modem2.speakerOn = False
        End If
    End Sub

    Private Sub CheckBox7_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox7.CheckedChanged
        If CheckBox7.Checked Then
            modem3.speakerOn = True
        Else
            modem3.speakerOn = False
        End If
    End Sub

    Private Sub CheckBox8_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox8.CheckedChanged
        If CheckBox8.Checked Then
            modem4.speakerOn = True
        Else
            modem4.speakerOn = False
        End If
    End Sub


    Private Sub ReEnableListen()
        'immediately re-enables listen during the work week
        Dim ttt As TimeSpan = Date.Now.TimeOfDay
        Dim ddd As DayOfWeek = Date.Now.DayOfWeek
        If ttt > appset.WorkdayStart AndAlso ttt < appset.WorkdayEnd Then
            Select Case ddd
                Case DayOfWeek.Sunday
                    'do nothing
                Case DayOfWeek.Saturday
                    If appset.IsSatAWorkday Then
                        If CheckBox1.Enabled Then
                            CheckBox1.Checked = True
                        End If
                        If CheckBox2.Enabled Then
                            CheckBox2.Checked = True
                        End If
                        If CheckBox3.Enabled Then
                            CheckBox3.Checked = True
                        End If
                        If CheckBox4.Enabled Then
                            CheckBox4.Checked = True
                        End If
                        Me.stampedTime = ttt
                    End If
                    'otherwise
                Case Else
                    If CheckBox1.Enabled Then
                        CheckBox1.Checked = True
                    End If
                    If CheckBox2.Enabled Then
                        CheckBox2.Checked = True
                    End If
                    If CheckBox3.Enabled Then
                        CheckBox3.Checked = True
                    End If
                    If CheckBox4.Enabled Then
                        CheckBox4.Checked = True
                    End If
                    Me.stampedTime = ttt
            End Select
        End If
    End Sub

#End Region

#Region "Menu Strip Controls"
    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

        Me.m_closeOK = True
        SaveLDUnitList()
        Application.Exit()

    End Sub

    Private Sub DownloadOneUnitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles DownloadOneUnitToolStripMenuItem.Click
        Form1.uv.Hide()
        Me.CmsForTrayIcon.Enabled = False
        Dim one As New OneUnitDialog()
        Dim dRes As DialogResult = one.ShowDialog(Me)
        Me.CmsForTrayIcon.Enabled = True

        Select Case dRes
            Case Windows.Forms.DialogResult.OK
                appset.DialInProg = True
                Dim EA As New StatusBarEventArgs(StatusBarEventArgs.SBAction.Populate)
                RaiseEvent SBUnitCounter(Me, EA)
                RaiseEvent UpdateMessages(Me, EA)

                If modem1.modemMode = Modem.Mode.Standby AndAlso Not BW1.IsBusy Then
                    txtOut1.Clear()
                    txtOut1h.Clear()
                    rtfBody1.Clear()
                    modem1.modemMode = Modem.Mode.Dial
                    BW1.RunWorkerAsync()
                End If
                If modem2.modemMode = Modem.Mode.Standby AndAlso Not BW2.IsBusy Then
                    txtOut2.Clear()
                    txtOut2h.Clear()
                    rtfBody2.Clear()
                    modem2.modemMode = Modem.Mode.Dial
                    BW2.RunWorkerAsync()
                End If
                If modem3.modemMode = Modem.Mode.Standby AndAlso Not BW3.IsBusy Then
                    txtOut3.Clear()
                    txtOut3h.Clear()
                    rtfBody3.Clear()
                    modem3.modemMode = Modem.Mode.Dial
                    BW3.RunWorkerAsync()
                End If
                If modem4.modemMode = Modem.Mode.Standby AndAlso Not BW4.IsBusy Then
                    txtOut4.Clear()
                    txtOut4h.Clear()
                    rtfBody4.Clear()
                    modem4.modemMode = Modem.Mode.Dial
```

```vb
                    BW4.RunWorkerAsync()
                End If
                If modem1.modemMode = Modem.Mode.Dial OrElse modem2.modemMode = Modem.Mode.Dial _
                OrElse modem3.modemMode = Modem.Mode.Dial OrElse modem4.modemMode = Modem.Mode.Dial Then
                    'a dial operation is in progress - display the Cancel Downloads button
                    appset.DialInProg = True
                    Button2.Text = "Cancel Downloads"
                    Button2_TextChanged(Me, e)
                End If

            Case Else
                'do nothing
        End Select

        If IsUVOpen Then
            Form1.uv.Show()
            Me.Focus()
        End If
    End Sub

    Private Sub SaveToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles SaveToolStripMenuItem.Click

        Form1.uv.Hide()
        SaveFileDialog1.InitialDirectory = "Desktop"
        SaveFileDialog1.FileName = "Log " & Date.Now.ToString("ddMMMyyyy HHmm")
        SaveFileDialog1.ShowDialog(Me)

        If Form1.IsUVOpen Then
            Form1.uv.Show()
            Me.Focus()
        End If
    End Sub
    Private Sub SaveFileDialog1_FileOk(ByVal sender As System.Object, ByVal e As System.ComponentModel. ↙
    CancelEventArgs) Handles SaveFileDialog1.FileOk

        Me.Cursor = Cursors.WaitCursor
        Me.CmsForTrayIcon.Enabled = False
        Me.SaveToolStripMenuItem.Enabled = False

        Dim textBoxSave As New Windows.Forms.RichTextBox

        textBoxSave.Multiline = True
        textBoxSave.ForeColor = Color.Blue
        textBoxSave.AppendText("Log file created " & Date.Now.ToString("M/d/yyyy HH:mm:ss") & vbLf & vbLf)

        textBoxSave.AppendText("Messages" & vbLf & vbLf)
        textBoxSave.Select(textBoxSave.TextLength, 0) 'sets caret at the end of the text
        textBoxSave.SelectedRtf = Me.rtbMessages.Rtf
        textBoxSave.Select(textBoxSave.TextLength, 0)

        textBoxSave.AppendText(vbLf & vbLf & vbLf & "Modem 1" & vbLf & vbLf & vbLf)
        textBoxSave.Select(textBoxSave.TextLength, 0)
        textBoxSave.SelectedRtf = Me.txtOut1h.Rtf
        textBoxSave.Select(textBoxSave.TextLength, 0)

        textBoxSave.AppendText(vbLf & vbLf & vbLf & "Modem 2" & vbLf & vbLf & vbLf)
        textBoxSave.Select(textBoxSave.TextLength, 0)
        textBoxSave.SelectedRtf = Me.txtOut2h.Rtf
        textBoxSave.Select(textBoxSave.TextLength, 0)

        textBoxSave.AppendText(vbLf & vbLf & vbLf & "Modem 3" & vbLf & vbLf & vbLf)
        textBoxSave.Select(textBoxSave.TextLength, 0)
        textBoxSave.SelectedRtf = Me.txtOut3h.Rtf
        textBoxSave.Select(textBoxSave.TextLength, 0)

        textBoxSave.AppendText(vbLf & vbLf & vbLf & "Modem 4" & vbLf & vbLf & vbLf)
        textBoxSave.Select(textBoxSave.TextLength, 0)
        textBoxSave.SelectedRtf = Me.txtOut4h.Rtf
        textBoxSave.Select(textBoxSave.TextLength, 0)

        textBoxSave.AppendText(vbLf & vbLf & "*** End of log ***" & vbLf & vbLf & vbLf)

        textBoxSave.SaveFile(SaveFileDialog1.FileName)

        'clear the hidden textboxes
        Me.txtOut1h.Clear()
        Me.txtOut2h.Clear()
        Me.txtOut3h.Clear()
        Me.txtOut4h.Clear()

        Me.Cursor = Cursors.Default
        Me.CmsForTrayIcon.Enabled = True
        Me.SaveToolStripMenuItem.Enabled = True
    End Sub

    Private Sub ViewUnitsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
     Handles ViewUnitsToolStripMenuItem.Click
        'opens the unit viewer - this window shouldn't be owned by Form1
        IsUVOpen = True
        If uv.Created Then
            uv.Show()
            If uv.WindowState = FormWindowState.Minimized Then
                uv.WindowState = FormWindowState.Normal
            End If
            uv.Focus()
        Else
            uv = New UnitView()
            uv.Show()
        End If

    End Sub

    Private Sub UnitOptionsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.        ↙
    EventArgs) Handles UnitOptionsToolStripMenuItem.Click

        Dim wasAnyoneListening As Boolean = False

        'cancel Listen operation,
        Me.wasListening(0) = CheckBox1.Checked
        Me.wasListening(1) = CheckBox2.Checked
        Me.wasListening(2) = CheckBox3.Checked
        Me.wasListening(3) = CheckBox4.Checked

        CheckBox1.Checked = False
        CheckBox2.Checked = False
        CheckBox3.Checked = False
        CheckBox4.Checked = False

        'If Not Me.BWDelayedUO.IsBusy Then
        '    BWDelayedUO.RunWorkerAsync()
        'End If

        For Each l As Boolean In wasListening
            If l Then
                wasAnyoneListening = True
            End If
        Next
        If wasAnyoneListening Then
            Do
                Me.Cursor = Cursors.WaitCursor
                Thread.Sleep(10)
                Application.DoEvents()
            Loop Until modem1.modemMode = Modem.Mode.Standby AndAlso modem2.modemMode = Modem.Mode.Standby _
            AndAlso modem3.modemMode = Modem.Mode.Standby AndAlso modem4.modemMode = Modem.Mode.Standby
            Me.Cursor = Cursors.Default
        End If

        'launch UnitOptions dialog in the UI thread
        Form1.uv.Hide()
        Me.CmsForTrayIcon.Enabled = False
        Dim uo As New UnitOptions()
        Dim dRes As DialogResult = uo.ShowDialog(Me)    'won't return to main form until this dialog box is ↙
    closed
        Me.CmsForTrayIcon.Enabled = True

        If dRes = Windows.Forms.DialogResult.OK Then
            SaveLDUnitList()
        End If

        're-launch Listen on the units that were listening
        Me.CheckBox1.Checked = Me.wasListening(0)
        Me.CheckBox2.Checked = Me.wasListening(1)
        Me.CheckBox3.Checked = Me.wasListening(2)
        Me.CheckBox4.Checked = Me.wasListening(3)

        If IsUVOpen Then
            Form1.uv.Show()
            Me.Focus() 'returns focus to Form1
        End If
    End Sub

    Private Sub ViewDatabaseToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.       ↙
    EventArgs) Handles ViewDatabaseToolStripMenuItem.Click

        Diagnostics.Process.Start("Database Viewer.exe")

    End Sub

    Private Sub OptionsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)  ↙
    Handles OptionsToolStripMenuItem.Click
        Form1.uv.Hide()
        Dim ass As New AppSettingsDialog()
        Dim dres As DialogResult = ass.ShowDialog()

        If dres = Windows.Forms.DialogResult.OK Then
            modem1.modemPhoneNum = appset.Modem1Num
            modem2.modemPhoneNum = appset.Modem2Num
            modem3.modemPhoneNum = appset.Modem3Num
            modem4.modemPhoneNum = appset.Modem4Num
        End If

        If IsUVOpen Then
            Form1.uv.Show()
```

```vb
            Me.Focus()
        End If
    End Sub

    Private Delegate Sub ChangeCursor()
    Private Sub normalCursor()
        Me.Cursor = Cursors.Default
    End Sub
    Private Sub waitCursor()
        Me.Cursor = Cursors.WaitCursor
    End Sub

#End Region

#Region "Tray Icon Controls"

    Private Sub RestoreToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RestoreToolStripMenuItem.Click
        Me.WindowState = wState
    End Sub

    Private Sub ManualDialToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ManualDialToolStripMenuItem.Click
        DownloadOneUnitToolStripMenuItem_Click(Me, e)
    End Sub

    Private Sub ExitLDToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitLDToolStripMenuItem.Click
        ExitToolStripMenuItem_Click(Me, e)
    End Sub

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Resize
        If Me.WindowState = FormWindowState.Minimized Then
            Me.NotifyIcon1.Visible = True
            Me.ShowInTaskbar = False
        Else
            Me.NotifyIcon1.Visible = False
            Me.ShowInTaskbar = True
            wState = Me.WindowState
        End If
    End Sub

    Private Sub NotifyIcon1_MouseDoubleClick(ByVal sender As System.Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles NotifyIcon1.MouseDoubleClick
        Me.WindowState = wState
    End Sub


    Private Sub SaveTextAsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaveTextAsToolStripMenuItem.Click

        Me.SaveToolStripMenuItem_Click(Me, New EventArgs())
    End Sub

    Private Sub ViewUnitsToolStripMenuItem1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ViewUnitsToolStripMenuItem1.Click

        Me.ViewUnitsToolStripMenuItem_Click(Me, New EventArgs())
    End Sub

    Private Sub UnitOptionsToolStripMenuItem1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles UnitOptionsToolStripMenuItem1.Click

        Me.UnitOptionsToolStripMenuItem_Click(Me, New EventArgs())
    End Sub

#End Region

    Private Sub Form1_FormClosingEventHandler(ByVal sender As Object, ByVal e As Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
        'idea from vb-helper.com

        If m_closeOK = False Then
            e.Cancel = True
            Me.WindowState = FormWindowState.Minimized
            Form1_Resize(Me, e)
        Else
            'SaveLDUnitList() moved to the Exit handler because this method is not called if Form1 is minimized
            'and the user exits the program
        End If
    End Sub



End Class

Public Class UpdateOutBoxesEventArgs
    Inherits EventArgs

    Private m_appendThisString As String
    Private m_RWFlag As RW

    Public Enum RW
        Read
        Write
        Notify
    End Enum

    Public Property AppendThisString() As String
        Get
            Return m_appendThisString
        End Get
        Set(ByVal value As String)
            m_appendThisString = value
        End Set
    End Property

    Public Property RWFlag() As RW
        Get
            Return m_RWFlag
        End Get
        Set(ByVal value As RW)
            m_RWFlag = value
        End Set
    End Property

    Public Sub New(ByVal value As String, ByVal flag As RW)
        m_appendThisString = value
        RWFlag = flag
    End Sub

End Class

Public Class StatusBarEventArgs
    Inherits EventArgs

    Private m_action As SBAction
    Private m_precedingText As String
    Private m_unitNum As String

    Public Property Action() As SBAction
        Get
            Return m_action
        End Get
        Set(ByVal value As SBAction)
            m_action = value
        End Set
    End Property
    Public Property PrecedingText() As String
        Get
            Return m_precedingText
        End Get
        Set(ByVal value As String)
            m_precedingText = value
        End Set
    End Property
    Public Property UnitNum() As String
        Get
            Return m_unitNum
        End Get
        Set(ByVal value As String)
            m_unitNum = value
        End Set
    End Property

    Public Enum SBAction
        Populate
        Add
        Remove
        Clear
        Cancelled
        ChangePrecedingText
        HideText
        ShowText
    End Enum

    Public Sub New(ByVal act As SBAction)   'use to clear or show the status bar text
        Action = act
        UnitNum = ""
    End Sub

    Public Sub New(ByVal unit As String, ByVal act As SBAction)
        UnitNum = unit
        Action = act
    End Sub



End Class

```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="WindowsApplication1.My.MySettings.ComplaintConnectionString"
            connectionString="Data Source=localhost;Initial Catalog=Complaint;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <system.diagnostics>
        <sources>
            <!-- This section defines the logging configuration for My.Application.Log -->
            <source name="DefaultSource" switchName="DefaultSwitch">
                <listeners>
                    <add name="FileLog"/>
                    <!-- Uncomment the below section to write to the Application Event Log -->
                    <!--<add name="EventLog"/>-->
                </listeners>
            </source>
        </sources>
        <switches>
            <add name="DefaultSwitch" value="Information" />
        </switches>
        <sharedListeners>
            <add name="FileLog"
                 type="Microsoft.VisualBasic.Logging.FileLogTraceListener, Microsoft.VisualBasic, Version=8.0.↙
  0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL"
                 initializeData="FileLogWriter"/>
            <!-- Uncomment the below section and replace APPLICATION_NAME with the name of your application to↙
   write to the Application Event Log -->
            <!--<add name="EventLog" type="System.Diagnostics.EventLogTraceListener" initializeData=↙
  "APPLICATION_NAME"/> -->
        </sharedListeners>
    </system.diagnostics>
</configuration>
```

Appendix B: Application Settings



Install Path:

Number of dial attempts before giving up:

Max number of lines to display in the main window:

Modem 1 Port Phone #

Modem 2 Port Phone #

Modem 3 Port Phone #

Modem 4 Port Phone #

☐ All modems have the same phone number

☐ Limit connection speed to 9600 bps (improves reliability)

When download is complete:

☐ Save log file

Note: Program must be restarted for port changes to take effect.
All other changes are effective immediately.

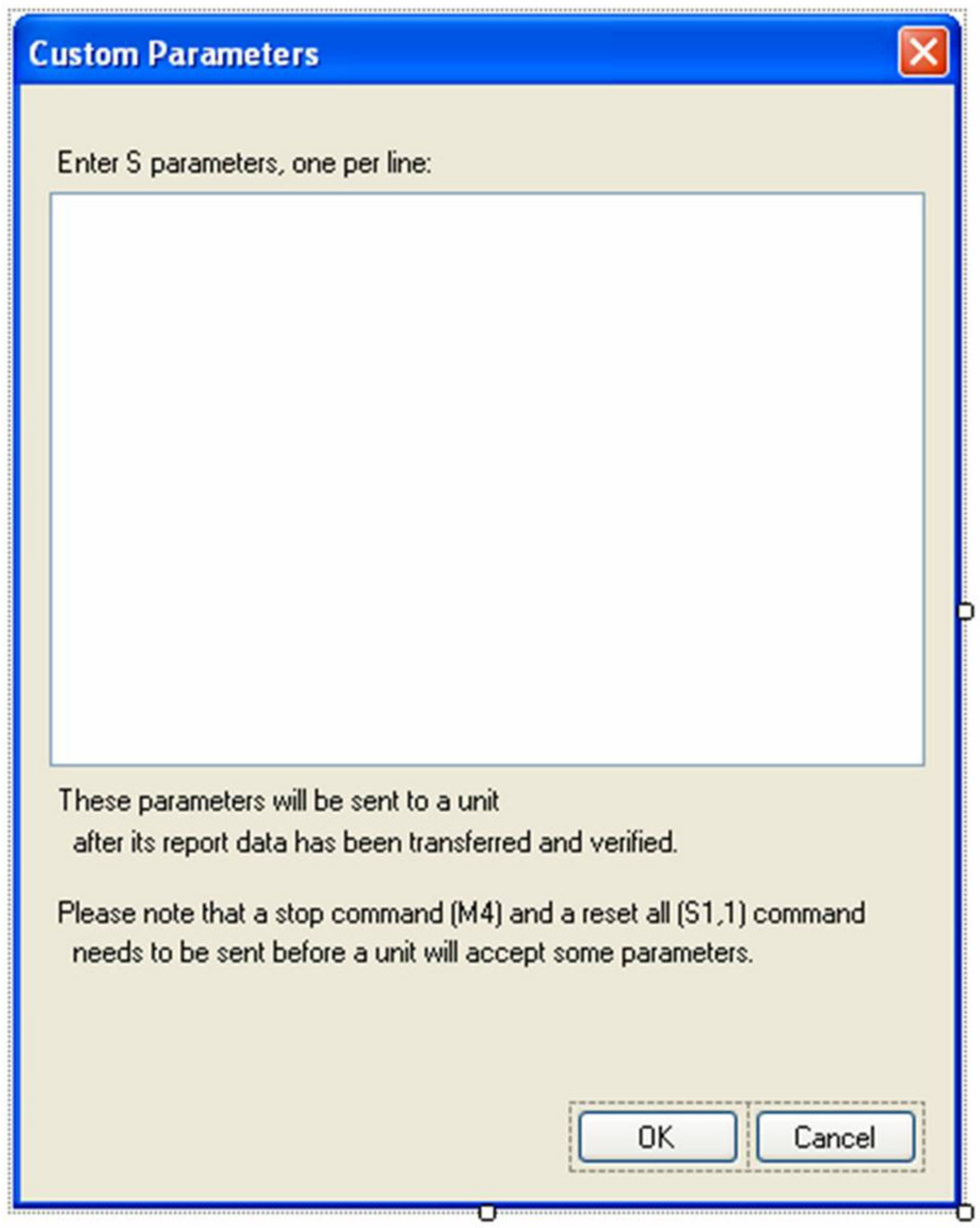
```vb
Imports System.Windows.Forms

Public Class AppSettingsDialog

    Private DA As Integer
    Private ints(11) As Integer
    Private excd1 As Integer
    Private excd2 As Integer
    Private mxLines As Integer
    Private closeOK As Boolean = True


    Private Sub OK_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles OK_Button↙
    .Click
        Dim modem1PortExists As Boolean = False
        Dim modem2PortExists As Boolean = False
        Dim modem3PortExists As Boolean = False
        Dim modem4PortExists As Boolean = False
        Dim noDuplicatePorts As Boolean = True

        For i As Integer = 0 To My.Computer.Ports.SerialPortNames.Count - 1
            If Me.cbbPort1.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                modem1PortExists = True
                If Me.cbbPort2.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort3.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort4.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                    noDuplicatePorts = False
                End If
            End If
            If Me.cbbPort2.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                modem2PortExists = True
                If Me.cbbPort1.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort3.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort4.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                    noDuplicatePorts = False
                End If
            End If
            If Me.cbbPort3.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                modem3PortExists = True
                If Me.cbbPort2.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort1.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort4.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                    noDuplicatePorts = False
                End If
            End If
            If Me.cbbPort4.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                modem4PortExists = True
                If Me.cbbPort2.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort3.Text = My.Computer.Ports.SerialPortNames.Item(i) OrElse _
                Me.cbbPort1.Text = My.Computer.Ports.SerialPortNames.Item(i) Then
                    noDuplicatePorts = False
                End If
            End If
        Next

        If noDuplicatePorts Then
            If modem1PortExists Then
                Form1.appset.Modem1Port = Me.cbbPort1.Text
            Else
                Me.closeOK = False
            End If
            If modem2PortExists Then
                Form1.appset.Modem2Port = Me.cbbPort2.Text
            Else
                Me.closeOK = False
            End If
            If modem3PortExists Then
                Form1.appset.Modem3Port = Me.cbbPort3.Text
            Else
                Me.closeOK = False
            End If
            If modem4PortExists Then
                Form1.appset.Modem4Port = Me.cbbPort4.Text
            Else
                Me.closeOK = False
            End If
        Else
            closeOK = False
        End If
        If Not closeOK Then
            MessageBox.Show("The modem's serial port name is entered incorrectly." & vbLf & _
            "Please select a port from the pull-down list.", "Warning", MessageBoxButtons.OK, _
            MessageBoxIcon.Warning)
        End If


        With Form1.appset
            .InstallPath = tbInst.Text
            .DialAttempts = DA - 1
            .Modem1Num = tbM1.Text
            .Modem2Num = tbM2.Text
            .Modem3Num = tbM3.Text
            .Modem4Num = tbM4.Text
            .MaxLines = mxLines
            .WeeknightDL = New TimeSpan(ints(0), ints(1), ints(2))
            .WeekendDL = New TimeSpan(ints(3), ints(4), ints(5))
            .WorkdayStart = New TimeSpan(ints(6), ints(7), ints(8))
            .WorkdayEnd = New TimeSpan(ints(9), ints(10), ints(11))
            .IsSatAWorkday = cbSat.Checked
            .IsSunAWorkday = cbSun.Checked
            .StopAtNight = Me.cbStopAtNight.Checked
            .SaveLogAfterDL = cbSaveLog.Checked
            .Conn9600 = Me.cb9600.Checked
            .WorkdayExcdThresh = excd1
            .OtherExcdThresh = excd2
            .Serialize()
        End With

        'set each unit to the new timer value, if needed
        Dim ulist As List(Of LDUnit) = Form1.GetAllUnitsNoMark
        For Each u As LDUnit In ulist
            If u.TimerRun1 <> ints(6).ToString("d2") & ":" & ints(7).ToString("d2") _
            OrElse u.TimerStop1 <> ints(9).ToString("d2") & ":" & ints(10).ToString("d2") Then
                'enter new start and stop values, tell Modem to update the unit
                u.TimerRun1 = ints(6).ToString("d2") & ":" & ints(7).ToString("d2")
                u.TimerStop1 = ints(9).ToString("d2") & ":" & ints(10).ToString("d2")
                u.TimerChanged = True
            End If
        Next
        Form1.ReturnAllUnits(ulist)

        If Me.closeOK Then
            Me.DialogResult = System.Windows.Forms.DialogResult.OK
            Me.Close()
        Else
            Me.AppSettingsDialog_Load(Me, New EventArgs)
        End If

    End Sub

    Private Sub Cancel_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles    ↙
    Cancel_Button.Click
        Me.DialogResult = System.Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub

    Private Sub AppSettingsDialog_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles    ↙
    MyBase.Load

        Me.DA = Form1.appset.DialAttempts
        DA += 1
        Me.tbInst.Text = Form1.appset.InstallPath
        Me.tbDial.Text = DA.ToString
        Me.tbLines.Text = Form1.appset.MaxLines.ToString 'this automatically sets me.mxLabel = maxLines
        Me.tbM1.Text = Form1.appset.Modem1Num
        Me.tbM2.Text = Form1.appset.Modem2Num
        Me.tbM3.Text = Form1.appset.Modem3Num
        Me.tbM4.Text = Form1.appset.Modem4Num
        Me.tb1.Text = Form1.appset.WeeknightDL.Hours.ToString("00")
        Me.tb2.Text = Form1.appset.WeeknightDL.Minutes.ToString("00")
        Me.tb3.Text = Form1.appset.WeeknightDL.Seconds.ToString("00")
        Me.tb4.Text = Form1.appset.WeekendDL.Hours.ToString("00")
        Me.tb5.Text = Form1.appset.WeekendDL.Minutes.ToString("00")
        Me.tb6.Text = Form1.appset.WeekendDL.Seconds.ToString("00")
        Me.tb7.Text = Form1.appset.WorkdayStart.Hours.ToString("00")
        Me.tb8.Text = Form1.appset.WorkdayStart.Minutes.ToString("00")
        Me.tb9.Text = Form1.appset.WorkdayStart.Seconds.ToString("00")
        Me.tb10.Text = Form1.appset.WorkdayEnd.Hours.ToString("00")
        Me.tb11.Text = Form1.appset.WorkdayEnd.Minutes.ToString("00")
        Me.tb12.Text = Form1.appset.WorkdayEnd.Seconds.ToString("00")
        Me.cbSat.Checked = Form1.appset.IsSatAWorkday
        Me.cbSun.Checked = Form1.appset.IsSunAWorkday
        Me.cbStopAtNight.Checked = Form1.appset.StopAtNight
        Me.cbSaveLog.Checked = Form1.appset.SaveLogAfterDL
        Me.cb9600.Checked = Form1.appset.Conn9600

        ints(0) = Form1.appset.WeeknightDL.Hours
        ints(1) = Form1.appset.WeeknightDL.Minutes
        ints(2) = Form1.appset.WeeknightDL.Seconds
        ints(3) = Form1.appset.WeekendDL.Hours
        ints(4) = Form1.appset.WeekendDL.Minutes
        ints(5) = Form1.appset.WeekendDL.Seconds
        ints(6) = Form1.appset.WorkdayStart.Hours
        ints(7) = Form1.appset.WorkdayStart.Minutes
        ints(8) = Form1.appset.WorkdayStart.Seconds
        ints(9) = Form1.appset.WorkdayEnd.Hours
        ints(10) = Form1.appset.WorkdayEnd.Minutes
        ints(11) = Form1.appset.WorkdayEnd.Seconds

        Me.cbSameNum.Checked = True

        'hide stuff to change excd threshold - it's been moved to Unit Options

        'populate the port list combo box(es)
```

```vb
        cbbPort1.Items.Clear()
        For i As Integer = 0 To My.Computer.Ports.SerialPortNames.Count - 1
            cbbPort1.Items.Add(My.Computer.Ports.SerialPortNames.Item(i))
            cbbPort2.Items.Add(My.Computer.Ports.SerialPortNames(i))
            cbbPort3.Items.Add(My.Computer.Ports.SerialPortNames(i))
            cbbPort4.Items.Add(My.Computer.Ports.SerialPortNames(i))
        Next
        Me.cbbPort1.Text = Form1.appset.Modem1Port
        Me.cbbPort2.Text = Form1.appset.Modem2Port
        Me.cbbPort3.Text = Form1.appset.Modem3Port
        Me.cbbPort4.Text = Form1.appset.Modem4Port

        tt.SetToolTip(Me.cb9600, "Tells the local modems to connect at 9600bps." & vbLf & _
    "Clear this checkmark to let modems connect at the modem's maximum speed.")
        tt.SetToolTip(Me.tbLines, "Sets the number of text lines to display in" & vbLf & _
       "each of the four main windows." & vbLf & "Decrease this number if the program is sluggish upon restore." _
       & vbLf & vbLf & "Does not affect how much text is written to the log file.")
        Me.closeOK = True
    End Sub



    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim dres As DialogResult = FolderBrowserDialog1.ShowDialog()
        If dres = Windows.Forms.DialogResult.OK Then
            tbInst.Text = FolderBrowserDialog1.SelectedPath
        End If
    End Sub

    Private Sub cbSameNum_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbSameNum.CheckedChanged
        If cbSameNum.Checked Then
            tbM2.Enabled = False
            tbM3.Enabled = False
            tbM4.Enabled = False
        Else
            tbM2.Enabled = True
            tbM3.Enabled = True
            tbM4.Enabled = True
        End If
    End Sub

    Private Sub tbM1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tbM1.TextChanged
        If cbSameNum.Checked Then
            tbM2.Text = tbM1.Text
            tbM3.Text = tbM1.Text
            tbM4.Text = tbM1.Text
        End If
    End Sub


    Private Sub tbDial_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) Handles tbDial.LostFocus
        Dim tryp As Boolean = Integer.TryParse(tbDial.Text, DA)

        If tryp = False OrElse DA < 1 Then
            MessageBox.Show("The number of dial attempts must be an integer greater than zero.", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
            DA = Form1.appset.DialAttempts
            DA += 1
            tbDial.Text = DA.ToString
            tbDial.Focus()
        End If
    End Sub

#Region "Schedule text boxes"
    Private Sub tb1_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb1.Enter
        tb1.SelectAll()
    End Sub


    Private Sub Tb1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb1.TextChanged
        Dim res As Integer
        If tb1.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb1.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 24 Then
                ints(0) = res
                tb2.Focus()
            Else
                tb1.Clear()
                tb1.Focus()
            End If
        End If
    End Sub
    Private Sub tb2_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb2.Enter
        tb2.SelectAll()
    End Sub
    Private Sub tb2_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb2.
    TextChanged
        Dim res As Integer
        If tb2.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb2.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(1) = res
                tb3.Focus()
            Else
                tb2.Clear()
                tb2.Focus()
            End If
        End If

    End Sub

    Private Sub tb3_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb3.Enter
        tb3.SelectAll()
    End Sub

    Private Sub tb3_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb3. ↙
    TextChanged
        Dim res As Integer
        If tb3.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb3.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(2) = res
                tb4.Focus()
            Else
                tb3.Clear()
                tb3.Focus()
            End If
        End If

    End Sub


    Private Sub tb4_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb4.Enter
        tb4.SelectAll()
    End Sub


    Private Sub tb5_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb5.Enter
        tb5.SelectAll()
    End Sub

    Private Sub tb6_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb6.Enter
        tb6.SelectAll()
    End Sub

    Private Sub tb7_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb7.Enter
        tb7.SelectAll()
    End Sub

    Private Sub tb8_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb8.Enter
        tb8.SelectAll()
    End Sub

    Private Sub tb9_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb9.Enter
        tb9.SelectAll()
    End Sub

    Private Sub tb10_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb10.Enter
        tb10.SelectAll()
    End Sub

    Private Sub tb11_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb11.Enter
        tb11.SelectAll()
    End Sub

    Private Sub tb12_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles tb12.Enter
        tb12.SelectAll()
    End Sub

    Private Sub tb4_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb4. ↙
    TextChanged
        Dim res As Integer
        If tb4.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb4.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 24 Then
                ints(3) = res
                tb5.Focus()
            Else
                tb4.Clear()
                tb4.Focus()
            End If
        End If
    End Sub

    Private Sub tb5_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb5. ↙
    TextChanged
        Dim res As Integer
        If tb5.Text.Length >= 2 Then
```

```vb
            Dim tryp As Boolean = Integer.TryParse(tb5.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(4) = res
                tb6.Focus()
            Else
                tb5.Clear()
                tb5.Focus()
            End If
        End If
    End Sub

    Private Sub tb6_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb6.TextChanged
        Dim res As Integer
        If tb6.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb6.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(5) = res
                tb7.Focus()
            Else
                tb6.Clear()
                tb6.Focus()
            End If
        End If
    End Sub

    Private Sub tb7_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb7.TextChanged
        Dim res As Integer
        If tb7.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb7.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 24 Then
                ints(6) = res
                tb8.Focus()
            Else
                tb7.Clear()
                tb7.Focus()
            End If
        End If
    End Sub

    Private Sub tb8_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb8.TextChanged
        Dim res As Integer
        If tb8.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb8.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(7) = res
                tb9.Focus()
            Else
                tb8.Clear()
                tb8.Focus()
            End If
        End If

    End Sub

    Private Sub tb9_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb9.TextChanged
        Dim res As Integer
        If tb9.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb9.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(8) = res
                tb10.Focus()
            Else
                tb9.Clear()
                tb9.Focus()
            End If
        End If
    End Sub

    Private Sub tb10_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb10.TextChanged
        Dim res As Integer
        If tb10.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb10.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 24 Then
                ints(9) = res
                tb11.Focus()
            Else
                tb10.Clear()
                tb10.Focus()
            End If
        End If

    End Sub

    Private Sub tb11_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb11.TextChanged
        Dim res As Integer
        If tb11.Text.Length >= 2 Then
```

```vb
            Dim tryp As Boolean = Integer.TryParse(tb11.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(10) = res
                tb12.Focus()
            Else
                tb11.Clear()
                tb11.Focus()
            End If
        End If
    End Sub

    Private Sub tb12_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tb12.TextChanged
        Dim res As Integer
        If tb12.Text.Length >= 2 Then
            Dim tryp As Boolean = Integer.TryParse(tb12.Text, res)
            If tryp AndAlso res >= 0 AndAlso res < 60 Then
                ints(11) = res
                tb12.Focus()
            Else
                tb12.Clear()
                tb12.Focus()
            End If
        End If
    End Sub

#End Region

    'Private Sub tbExcd1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    '    'makes sure the text entered is a number between 0 and 1000
    '    Dim res As Integer
    '    Dim tryp As Boolean = Integer.TryParse(tbExcd1.Text, res)

    '    If tryp = True AndAlso res > -1 AndAlso res < 1000 Then
    '        Me.excd1 = res
    '    End If
    'End Sub

    'Private Sub tbExcd2_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    '    'makes sure the text entered is a number between 0 and 1000
    '    Dim res As Integer
    '    Dim tryp As Boolean = Integer.TryParse(tbExcd2.Text, res)

    '    If tryp = True AndAlso res > -1 AndAlso res < 1000 Then
    '        Me.excd2 = res
    '    End If
    'End Sub

    Private Sub tbLines_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tbLines.TextChanged
        'makes sure the text entered is greater than zero
        Dim res As Integer
        Dim tryp As Boolean = Integer.TryParse(tbLines.Text, res)

        If tryp = True AndAlso res > 19 Then
            Me.mxLines = res
        End If
    End Sub


End Class


```

Appendix C: Custom Parameters



Enter S parameters, one per line:

These parameters will be sent to a unit
after its report data has been transferred and verified.

Please note that a stop command (M4) and a reset all (S1,1) command
needs to be sent before a unit will accept some parameters.

```
Imports System.Windows.Forms

Public Class CustomParamsDialog

    Private Sub OK_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.
    Click
        Me.DialogResult = System.Windows.Forms.DialogResult.OK
        Me.Close()
    End Sub

    Private Sub Cancel_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    Cancel_Button.Click
        Me.DialogResult = System.Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub

    Private Sub CustomParamsDialog_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MyBase.Load
        Me.TextBox1.Focus()

    End Sub
End Class

```

```
'LDUnit

'an object which represents one LD unit

Imports System
Imports System.IO
Imports System.IO.Ports
'Imports System.Text
Imports System.Threading
Imports System.Data.SqlClient

<Serializable()> _
Public Class LDUnit
    'path to which to serialize objects
    Protected m_installPath As String

    'report printing parameters
    'The UI and scheduler will set these properties:

    Protected m_includeR As Boolean
    Protected m_includeLC As Boolean
    Protected m_includeQ As Boolean
    Protected m_allowCallIns As Boolean
    Protected m_resetDataYN As Boolean
    Protected m_resetTimeYN As Boolean
    Protected m_isDownloadDone As Boolean
    Protected m_isDownloadInProgress As Boolean
    Protected m_isEnabled As Boolean
    Protected m_dlTries As Integer
    Protected m_dialTries As Integer
    Protected m_initialize As Boolean
    Protected m_sendCustP As Boolean

    '   Unit serial number
    '   Unit symbolic "L" number
    '   Unit password (11111111 or 22222222)
    'unit location (helps in identifying it)
    'unit phone number

    Protected m_unitSerial As String = ""
    Protected m_unitLNum As String = ""
    Protected m_lockCode As String = ""
    Protected m_unitLocation As String = ""
    Protected m_unitPhoneNum As String = ""
    Protected m_unitOwner As Owner
    Protected m_lastDL As DateTime
    Protected m_custParams As String = ""

    'These are R and Q parameters read from the unit upon download
    Protected m_numExceedances As Integer = -1
    Protected m_numIntervals As Integer = -1
    Protected m_numStartStops As Integer = -1
    Protected m_numCalibrations As Integer = -1
    Protected m_battVoltage As Double = -1.1
    Protected m_errorString As String = ""
    Protected m_eList As New List(Of String)
    Protected m_iList As New List(Of String)
    Protected m_rList As New List(Of String)
    Protected m_qList As New List(Of String)
    Protected m_lList As New List(Of String)
    Protected m_cList As New List(Of String)
    Protected m_excdThreshold As Integer = -1
    Protected m_calLevel As Double = -1
    Protected m_excdThreshDay As Integer = -1
    Protected m_excdThreshNight As Integer = -1
    Protected m_timerRun1 As String = ""
    Protected m_timerStop1 As String = ""
    Protected m_timerChanged As Boolean


    Public Enum Owner
        Aberdeen
        CERL
        Nobody
    End Enum


#Region "Properties"
    Public Property InstallPath() As String
        Get
            Return m_installPath
        End Get
        Set(ByVal value As String)
            m_installPath = value
        End Set
    End Property

    Public Property IncludeR() As Boolean
        Get
            Return m_includeR
        End Get
        Set(ByVal value As Boolean)
```

```
            m_includeR = value
        End Set
    End Property
    Public Property IncludeLC() As Boolean
        Get
            Return m_includeLC
        End Get
        Set(ByVal value As Boolean)
            m_includeLC = value
        End Set
    End Property
    Public Property IncludeQ() As Boolean
        Get
            Return m_includeQ
        End Get
        Set(ByVal value As Boolean)
            m_includeQ = value
        End Set
    End Property
    Public Property AllowCallIns() As Boolean
        Get
            Return m_allowCallIns
        End Get
        Set(ByVal value As Boolean)
            m_allowCallIns = value
        End Set
    End Property
    Public Property ResetDataYN() As Boolean
        Get
            Return m_resetDataYN
        End Get
        Set(ByVal value As Boolean)
            m_resetDataYN = value
        End Set
    End Property
    Public Property ResetTimeYN() As Boolean
        Get
            Return m_resetTimeYN
        End Get
        Set(ByVal value As Boolean)
            m_resetTimeYN = value
        End Set
    End Property
    Public Property IsDownloadDone() As Boolean
        Get
            Return m_isDownloadDone
        End Get
        Set(ByVal value As Boolean)
            m_isDownloadDone = value
        End Set
    End Property
    Public Property IsDownloadInProgress() As Boolean
        Get
            Return m_isDownloadInProgress
        End Get
        Set(ByVal value As Boolean)
            m_isDownloadInProgress = value
        End Set
    End Property
    Public Property IsEnabled() As Boolean
        Get
            Return m_isEnabled
        End Get
        Set(ByVal value As Boolean)
            m_isEnabled = value
        End Set
    End Property
    Public Property DLTries() As Integer
        Get
            Return m_dlTries
        End Get
        Set(ByVal value As Integer)
            m_dlTries = value
        End Set
    End Property
    Public Property DialTries() As Integer
        Get
            Return m_dialTries
        End Get
        Set(ByVal value As Integer)
            m_dialTries = value
        End Set
    End Property
    Public Property Initialize() As Boolean
        Get
            Return m_initialize
        End Get
        Set(ByVal value As Boolean)
            m_initialize = value
        End Set
    End Property
    Public Property SendCustP() As Boolean
```

```
        Get
            Return m_sendCustP
        End Get
        Set(ByVal value As Boolean)
            m_sendCustP = value
        End Set
    End Property


    Public Property UnitSerial() As String
        Get
            Return m_unitSerial
        End Get
        Set(ByVal value As String)
            m_unitSerial = value
        End Set
    End Property
    Public Property UnitNum() As String
        Get
            Return m_unitLNum
        End Get
        Set(ByVal value As String)
            m_unitLNum = value
        End Set
    End Property
    Public Property LockCode() As String
        Get
            Return m_lockCode
        End Get
        Set(ByVal value As String)
            m_lockCode = value
        End Set
    End Property
    Public Property UnitLocation() As String
        Get
            Return m_unitLocation
        End Get
        Set(ByVal value As String)
            m_unitLocation = value
        End Set
    End Property
    Public Property UnitPhoneNum() As String
        Get
            Return m_unitPhoneNum
        End Get
        Set(ByVal value As String)
            m_unitPhoneNum = value
        End Set
    End Property
    Public Property UnitOwner() As Owner
        Get
            Return m_unitOwner
        End Get
        Set(ByVal value As Owner)
            m_unitOwner = value
        End Set
    End Property
    Public Property LastDL() As DateTime
        Get
            Return m_lastDL
        End Get
        Set(ByVal value As DateTime)
            m_lastDL = value
        End Set
    End Property
    Public Property CustParams() As String
        Get
            Return m_custParams
        End Get
        Set(ByVal value As String)
            m_custParams = value
        End Set
    End Property


    Public Property NumExceedances() As Integer
        Get
            Return m_numExceedances
        End Get
        Set(ByVal value As Integer)
            m_numExceedances = value
        End Set
    End Property
    Public Property NumIntervals() As Integer
        Get
            Return m_numIntervals
        End Get
        Set(ByVal value As Integer)
            m_numIntervals = value
        End Set
    End Property
    Public Property NumStartStops() As Integer
```

```
        Get
            Return m_numStartStops
        End Get
        Set(ByVal value As Integer)
            m_numStartStops = value
        End Set
    End Property
    Public Property NumCalibrations() As Integer
        Get
            Return m_numCalibrations
        End Get
        Set(ByVal value As Integer)
            m_numCalibrations = value
        End Set
    End Property
    Public Property BattVoltage() As Double
        Get
            Return m_battVoltage
        End Get
        Set(ByVal value As Double)
            m_battVoltage = value
        End Set
    End Property
    Public Property ErrorString() As String
        Get
            Return m_errorString
        End Get
        Set(ByVal value As String)
            m_errorString = value
        End Set
    End Property
    Public Property EList() As List(Of String)
        Get
            Return m_eList
        End Get
        Set(ByVal value As List(Of String))
            m_eList = value
        End Set
    End Property
    Public Property IList() As List(Of String)
        Get
            Return m_iList
        End Get
        Set(ByVal value As List(Of String))
            m_iList = value
        End Set
    End Property
    Public Property RList() As List(Of String)
        Get
            Return m_rList
        End Get
        Set(ByVal value As List(Of String))
            m_rList = value
        End Set
    End Property
    Public Property QList() As List(Of String)
        Get
            Return m_qList
        End Get
        Set(ByVal value As List(Of String))
            m_qList = value
        End Set
    End Property
    Public Property LList() As List(Of String)
        Get
            Return m_lList
        End Get
        Set(ByVal value As List(Of String))
            m_lList = value
        End Set
    End Property
    Public Property CList() As List(Of String)
        Get
            Return m_cList
        End Get
        Set(ByVal value As List(Of String))
            m_cList = value
        End Set
    End Property
    Public Property ExcdThreshold() As Integer
        Get
            Return m_excdThreshold
        End Get
        Set(ByVal value As Integer)
            m_excdThreshold = value
        End Set
    End Property
    Public Property CalLevel() As Double
        Get
            Return m_calLevel
        End Get
        Set(ByVal value As Double)
```

```
            m_calLevel = value
        End Set
    End Property
    Public Property ExcdDay() As Integer
        Get
            Return m_excdThreshDay
        End Get
        Set(ByVal value As Integer)
            m_excdThreshDay = value
        End Set
    End Property
    Public Property ExcdNight() As Integer
        Get
            Return m_excdThreshNight
        End Get
        Set(ByVal value As Integer)
            m_excdThreshNight = value
        End Set
    End Property
    Public Property TimerRun1() As String
        Get
            Return m_timerRun1
        End Get
        Set(ByVal value As String)
            m_timerRun1 = value
        End Set
    End Property
    Public Property TimerStop1() As String
        Get
            Return m_timerStop1
        End Get
        Set(ByVal value As String)
            m_timerStop1 = value
        End Set
    End Property
    Public Property TimerChanged() As Boolean
        Get
            Return m_timerChanged
        End Get
        Set(ByVal value As Boolean)
            m_timerChanged = value
        End Set
    End Property


#End Region

#Region "Constructors"
    Public Sub New()
        Me.InstallPath = "C:\Program Files\L-D Download\"
        Me.IncludeR = False
        Me.IncludeLC = False
        Me.IncludeQ = False
        Me.AllowCallIns = False
        Me.ResetDataYN = False
        Me.ResetTimeYN = False
        Me.IsDownloadDone = False
        Me.IsDownloadInProgress = False
        Me.IsEnabled = True
        Me.DLTries = 0
        Me.DialTries = 0
        Me.Initialize = False
        Me.SendCustP = False
        Me.UnitLocation = ""
        Me.UnitNum = "null"
        Me.LastDL = New Date(1900, 1, 1, 0, 0, 0)
        Me.ExcdDay = Form1.appset.WorkdayExcdThresh
        Me.ExcdNight = Form1.appset.OtherExcdThresh
        Me.TimerRun1 = "06:00"
        Me.TimerStop1 = "22:00"
        Me.TimerChanged = False
    End Sub
    Public Sub New(ByVal unitnum As String)
        Me.InstallPath = "C:\Program Files\L-D Download\"
        Me.IncludeR = False
        Me.IncludeLC = False
        Me.IncludeQ = False
        Me.AllowCallIns = False
        Me.ResetDataYN = False
        Me.ResetTimeYN = False
        Me.IsDownloadDone = False
        Me.IsDownloadInProgress = False
        Me.IsEnabled = True
        Me.DLTries = 0
        Me.DialTries = 0
        Me.Initialize = False
        Me.SendCustP = False
        Me.UnitNum = unitnum
        Me.UnitLocation = ""
        Me.UnitOwner = Owner.Nobody
        Me.LastDL = New Date(1900, 1, 1, 0, 0, 0)
        Select Case Me.UnitOwner
```

```vb
                Case Owner.Aberdeen : Me.LockCode = "22222222"
                Case Owner.CERL : Me.LockCode = "22222222"
                Case Owner.Nobody : Me.LockCode = "22222222"
            End Select
            Me.ExcdDay = Form1.appset.WorkdayExcdThresh
            Me.ExcdNight = Form1.appset.OtherExcdThresh
            Me.TimerRun1 = "06:00"
            Me.TimerStop1 = "22:00"
            Me.TimerChanged = False
        End Sub
        Public Sub New(ByVal unitnum As String, ByVal unitphonenum As String, ByVal uowner As Owner)
            Me.InstallPath = "C:\Program Files\L-D Download\"
            Me.IncludeR = False
            Me.IncludeLC = False
            Me.IncludeQ = False
            Me.AllowCallIns = False
            Me.ResetDataYN = False
            Me.ResetTimeYN = False
            Me.IsDownloadDone = False
            Me.IsDownloadInProgress = False
            Me.IsEnabled = True
            Me.DLTries = 0
            Me.DialTries = 0
            Me.Initialize = False
            Me.SendCustP = False
            Me.UnitNum = unitnum
            Me.UnitLocation = ""
            Me.UnitPhoneNum = unitphonenum
            Me.UnitOwner = uowner
            Me.LastDL = New Date(1900, 1, 1, 0, 0, 0)
            Select Case Me.UnitOwner
                Case Owner.Aberdeen : Me.LockCode = "22222222"
                Case Owner.CERL : Me.LockCode = "22222222"
                Case Owner.Nobody : Me.LockCode = "22222222"
            End Select
            Me.ExcdDay = Form1.appset.WorkdayExcdThresh
            Me.ExcdNight = Form1.appset.OtherExcdThresh
            Me.TimerRun1 = "06:00"
            Me.TimerStop1 = "22:00"
            Me.TimerChanged = False
        End Sub
        Public Sub New(ByVal unitnum As String, ByVal unitphonenum As String, ByVal uowner As Owner, ByVal uLoc ↙
        As String)
            Me.InstallPath = "C:\Program Files\L-D Download\"
            Me.IncludeR = False
            Me.IncludeLC = False
            Me.IncludeQ = False
            Me.AllowCallIns = False
            Me.ResetDataYN = False
            Me.ResetTimeYN = False
            Me.IsDownloadDone = False
            Me.IsDownloadInProgress = False
            Me.IsEnabled = True
            Me.DLTries = 0
            Me.DialTries = 0
            Me.Initialize = False
            Me.SendCustP = False
            Me.UnitNum = unitnum
            Me.UnitLocation = ""
            Me.UnitPhoneNum = unitphonenum
            Me.UnitOwner = uowner
            Me.UnitLocation = uLoc
            Me.LastDL = New Date(1900, 1, 1, 0, 0, 0)
            Select Case Me.UnitOwner
                Case Owner.Aberdeen : Me.LockCode = "22222222"
                Case Owner.CERL : Me.LockCode = "22222222"
                Case Owner.Nobody : Me.LockCode = "22222222"
            End Select
            Me.ExcdDay = Form1.appset.WorkdayExcdThresh
            Me.ExcdNight = Form1.appset.OtherExcdThresh
            Me.TimerRun1 = "06:00"
            Me.TimerStop1 = "22:00"
            Me.TimerChanged = False
        End Sub
    #End Region

    #Region "Methods"

        Public Function SerializeReferenceTime(ByVal u1 As Date, ByVal h1 As Date, ByVal o1 As Double) As String
            'exports the time data taken when the unit was downloaded
            'stores the data as a serialized object

            Dim RT As ReferenceTime = New ReferenceTime(u1, h1, o1)

            Using fs As New FileStream(Me.InstallPath & "Reference Times\" & _
            Me.UnitNum & ".dat", FileMode.Create)

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                bf.Serialize(fs, RT)
            End Using

            'return the unique part of the filename
            Return Me.UnitNum & ".dat" '& "_" & h1.ToLongDateString

        End Function

        Public Function SerializeReferenceTime(ByVal RT As ReferenceTime) As String
            'exports the time data taken when the unit was downloaded
            'stores the data as a serialized object

            Using fs As New FileStream(Me.InstallPath & "Reference Times\" & _
            Me.UnitNum & ".dat", FileMode.Create)

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                bf.Serialize(fs, RT)
            End Using

            'return the unique part of the filename
            Return Me.UnitNum & ".dat" '& "_" & RT.HostTimeZero.ToLongDateString

        End Function

        Public Function GetReferenceTime() As ReferenceTime
            'imports a serialized object which contains the previously recorded time
            Dim RT As ReferenceTime

            Try
                Using fs As New FileStream(Me.InstallPath & "Reference Times\" & Me.UnitNum _
                        & ".dat", FileMode.Open)

                    Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                    RT = DirectCast(bf.Deserialize(fs), ReferenceTime)

                End Using
            Catch ex As FileNotFoundException   'file does not exist
                RT = New ReferenceTime(New Date(1), New Date(1), 0)

            End Try

            Return RT

        End Function

        Public Sub SerializeReports()
            'Using sw As New StreamWriter(Me.InstallPath & "Reports\" & Me.UnitNum & " " & _
            'Date.Now.ToString("dd-MMM-yy HH_mm_ss") & ".txt", True)

            '    For Each item As String In EList
            '        sw.WriteLine(item)
            '    Next
            '    For Each item As String In IList
            '        sw.WriteLine(item)
            '    Next


            '    sw.WriteLine()
            '    sw.WriteLine(BattVoltage & " V")
            '    sw.WriteLine(ErrorString)
            '    sw.WriteLine()
            '    sw.WriteLine("-------------------------------------------------------------------------")

            'End Using
            If Me.IncludeR OrElse Me.IncludeQ OrElse Me.IncludeLC Then
                Using sw2 As New StreamWriter(Me.InstallPath & "Setup Logs\" & Me.UnitNum & " Setup " & _
                Date.Now.ToString("dd-MMM-yy HH_mm_ss") & ".txt", True)
                    For Each item As String In RList
                        sw2.WriteLine(item)
                    Next
                    'For Each item As String In QList
                    '    sw2.WriteLine(item)
                    'Next
                    For Each item As String In LList
                        sw2.WriteLine(item)
                    Next
                    For Each item As String In CList
                        sw2.WriteLine(item)
                    Next

                End Using
            End If
        End Sub

        Public Function SaveToDB(ByVal RT As ReferenceTime) As String
            'saves report info to a database
            'returns an error string if an SQL error occurred

            Dim conn As New SqlConnection("server=localhost; database=Complaint; Integrated Security=SSPI")
            Dim sep() As Char = {","c}
            Dim sep2() As Char = {" "c}
            Dim trim1() As Char = {"'"c}
            Dim eCols As String = "UnitID,Date_Time,Duration,CLeq,CSEL,CLmax,CPeak,UwPeak,Counts,[Load],Comments↙
        ,Symmetry,Wind_Speed,Max_Wind_Speed,Wind_Direction,Temperature,Max_Temperature,Relative_Humidity,     ↙
        Max_Relative_Humidity"
            Dim iCols As String = "UnitID,Date_Time,Interval_End_Time,CLeq,CSEL,CLmin,CLmax,CPeak,UwPeak,       ↙
        Counts_a,Counts_b,Counts_c,Unknown,[L(5)],[L(5)_value],[L(10)],[L(10)_value],[L(33)],[L(33)_value],[L ↙
        (50)],[L(50)_value],[L(67)],[L(67)_value],[L(90)],[L(90)_value],Average_Wind_Speed,Max_Wind_Speed,     ↙
        Wind_Direction,Average_Temperature,Min_Temperature,Max_Temperature,Average_Relative_Humidity,           ↙
        Min_Relative_Humidity,Max_Relative_Humidity"

            Dim ctCol As String = ",Corrected_Time"
            Dim hrCol As String = ",TotalHrs"
            Dim hrCCol As String = ",TotalHrs_Corrected"
            Dim cmd As New SqlCommand("", conn)
            Dim errorStr As String = ""
            'Dim qCols As New System.Text.StringBuilder
            Dim qRow As New System.Text.StringBuilder
            Dim tryp1 As Boolean = False
            Dim tryp2 As Boolean = False

            Try
                conn.Open()
                For Each row As String In Me.EList
                    Dim cells() As String = row.Split(sep, StringSplitOptions.None)
                    row = ""
                    'cells will have 20 elements if corrected time is included, 19 if not
                    '(they have been error-checked by now)

                    'replace "E   1" with the unit number
                    cells(0) = Me.UnitNum

                    For i As Integer = 0 To cells.Length - 1
                        cells(i) = cells(i).Trim()
                        If i = 0 OrElse i = 1 OrElse i = 2 OrElse i = 10 OrElse i = 14 OrElse i = 19 Then
                            cells(i) = "'" & cells(i) & "'"
                        End If
                        row &= cells(i)
                        If i < cells.Length - 1 Then
                            row &= ", "
                        End If
                    Next

                    'append Total Hours and Corrected Total Hrs to the row string
                    'Dim dt As Date = Date.Now
                    'cells(1) = cells(1).Trim(trim1)
                    'tryp1 = Date.TryParse(cells(1), dt)
                    'If tryp1 Then
                    '    Dim dt2 As New TimeSpan(dt.Ticks)
                    '    Dim hrs As Double = dt2.TotalHours
                    '    hrs = Math.Round(hrs, 2)
                    '    row &= ", " & hrs.ToString
                    'End If

                    'If cells.Length = 20 Then 'append the total # of hours for the corrected time as well
                    '    cells(19) = cells(19).Trim(trim1)
                    '    tryp2 = Date.TryParse(cells(19), dt)
                    '    If tryp2 Then
                    '        Dim dt2 As New TimeSpan(dt.Ticks)
                    '        Dim hrs As Double = dt2.TotalHours
                    '        hrs = Math.Round(hrs, 2)
                    '        row &= ", " & hrs.ToString
                    '    End If
                    'End If

                    'MessageBox.Show(row)

                    'make the query string
                    If cells.Length = 20 Then
                        cmd.CommandText = "INSERT INTO EventData (" & eCols & ctCol & ") VALUES (" & row & ")"
                    Else
                        cmd.CommandText = "INSERT INTO EventData (" & eCols & ") VALUES (" & row & ")"
                    End If

                    'If cells.Length = 20 Then
                    '    If tryp1 AndAlso tryp2 Then
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & ctCol & hrCol & hrCCol & _
                    '                          ") VALUES (" & row & ")"
                    '    ElseIf tryp2 Then
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & ctCol & hrCCol & _
                    '                          ") VALUES (" & row & ")"
                    '    ElseIf tryp1 Then
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & ctCol & hrCol & _
                    '                          ") VALUES (" & row & ")"
                    '    Else
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & ctCol & _
                    '                          ") VALUES (" & row & ")"
                    '    End If
                    'Else
                    '    If tryp1 Then
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & hrCol & ") VALUES (" & row & ↙
        ")"
                    '    Else
                    '        cmd.CommandText = "INSERT INTO EventData (" & eCols & ") VALUES (" & row & ")"
                    '    End If
                    'End If

                    cmd.ExecuteNonQuery()
                Next

                For Each row As String In IList
                    Dim cells() As String = row.Split(sep, StringSplitOptions.None)
                    row = ""
                    'cells will have 35 elements if corrected time is included, 34 if not
                    '(they have been error-checked by now)

                    'replace "I   1" with the unit number
                    cells(0) = Me.UnitNum

                    For i As Integer = 0 To cells.Length - 1
                        cells(i) = cells(i).Trim()
                        If i = 0 OrElse i = 1 OrElse i = 2 OrElse i = 27 OrElse i = 34 Then
                            cells(i) = "'" & cells(i) & "'"
                        End If
                        row &= cells(i)
                        If i < cells.Length - 1 Then
                            row &= ", "
                        End If
                    Next

                    'append Total Hours and Corrected Total Hrs to the row string
                    'Dim dt As Date = Date.Now
                    'cells(1) = cells(1).Trim(trim1)
                    'tryp1 = Date.TryParse(cells(1), dt)
                    'If tryp1 Then
                    '    Dim dt2 As New TimeSpan(dt.Ticks)
                    '    Dim hrs As Double = dt2.TotalHours
                    '    hrs = Math.Round(hrs, 2)
                    '    row &= ", " & hrs.ToString
                    'End If

                    'If cells.Length = 35 Then 'append the total # of hours for the corrected time as well
                    '    cells(34) = cells(34).Trim(trim1)
                    '    tryp2 = Date.TryParse(cells(34), dt)
                    '    If tryp2 Then
                    '        Dim dt2 As New TimeSpan(dt.Ticks)
                    '        Dim hrs As Double = dt2.TotalHours
                    '        hrs = Math.Round(hrs, 2)
                    '        row &= ", " & hrs.ToString
                    '    End If
                    'End If

                    'make the query string
                    If cells.Length = 35 Then
                        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ctCol & ") VALUES (" & row & ")"
                    Else
                        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ") VALUES (" & row & ")"
                    End If

                    'If cells.Length = 35 Then
                    '    If tryp1 AndAlso tryp2 Then
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ctCol & hrCol & hrCCol & _
                    '                          ") VALUES (" & row & ")"
                    '    ElseIf tryp2 Then
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ctCol & hrCCol & _
                    '                          ") VALUES (" & row & ")"
                    '    ElseIf tryp1 Then
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ctCol & hrCol & _
                    '                          ") VALUES (" & row & ")"
                    '    Else
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ctCol & ") VALUES (" & row & ")"
                    '    End If
                    'Else
                    '    If tryp1 Then
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & hrCol & ") VALUES (" & row & ")"
                    '    Else
                    '        cmd.CommandText = "INSERT INTO Intervals (" & iCols & ") VALUES (" & row & ")"
                    '    End If
                    'End If

                    cmd.ExecuteNonQuery()
                Next

                'save Q's to Q table - entire setup file becomes one row
                If Me.IncludeQ Then
                    qRow.Append("'" & Me.UnitNum & "', '" & RT.HostTimeZero & "', ")
                    For Each line As String In QList
                        Dim cells() As String = line.Split(sep, StringSplitOptions.None)
                        Dim datacell As String = ""
                        'Dim qcell As String = ""
                        'cells(0) = cells(0).Trim()
                        ''If there are additional commas on this line, combine the different cells into one data line
                        For j As Integer = 1 To cells.Length - 1
                            datacell &= cells(j)
                        Next
                        cells(1) = datacell
                        cells(1) = cells(1).Trim()

                        'remove spaces between "Q  2"

                        'Dim Q() As String = cells(0).Split(sep2, StringSplitOptions.RemoveEmptyEntries)
                        'For j As Integer = 0 To Q.Length - 1
                        '    qcell &= Q(j)
                        'Next
                        'qCols.Append(cells(0) & ",")
                        qRow.Append("'" & cells(1) & "', ")
                    Next
                    'qCols.Remove(qCols.Length - 1, 1)
                    qRow.Remove(qRow.Length - 2, 2)

                    'append Total Hours to the corrected string
                    'Dim dt2 As New TimeSpan(RT.HostTimeZero.Ticks)
                    'Dim hrs As Double = dt2.TotalHours
                    'hrs = Math.Round(hrs, 2)
                    'qRow.Append("," & hrs.ToString)

                    'make the query string
                    cmd.CommandText = "INSERT INTO Q_Params VALUES (" & qRow.ToString() & ")"
                    'OK to not put column names - the RecordNum identifier gets skipped over automatically

                    cmd.ExecuteNonQuery()
                End If

                'record the new # of E rows to the table DB_Size
                'this is needed to make the database viewer update more efficiently

                cmd.CommandText = "SELECT count([Load]) FROM EventData"
                Dim numEDRows As Long = CLng(cmd.ExecuteScalar())
                cmd.CommandText = "UPDATE DB_Size SET NumEDRows = " & numEDRows.ToString & " WHERE ID = 1"
                cmd.ExecuteNonQuery()

            Catch ex As SqlException
                errorStr = "SQL Error: " & ex.ErrorCode & " - " & ex.Message
            Finally
                conn.Close()
            End Try

            Return errorStr

        End Function
    #End Region

    End Class



    <Serializable()> _
    Public Class ReferenceTime
        Protected m_unitTimeAvg1 As Date
        Protected m_hostTimeAvg1 As Date
        Protected m_tOffsetAvg1 As Double


        Public Property UnitTimeZero() As Date
            Get
                Return m_unitTimeAvg1
            End Get
            Set(ByVal value As Date)
                m_unitTimeAvg1 = value
            End Set
        End Property
        Public Property HostTimeZero() As Date
            Get
                Return m_hostTimeAvg1
            End Get
            Set(ByVal value As Date)
                m_hostTimeAvg1 = value
            End Set
        End Property
        Public Property TOffsetAvg1() As Double
            Get
                Return m_tOffsetAvg1
            End Get
            Set(ByVal value As Double)
                m_tOffsetAvg1 = value
            End Set
        End Property


        Public Sub New(ByVal u1 As Date, ByVal h1 As Date, ByVal o1 As Double)

            Me.UnitTimeZero = u1
            Me.HostTimeZero = h1
            Me.TOffsetAvg1 = o1
        End Sub

    End Class

    <Serializable()> _
```

```
Public Class AppSettings

    Protected m_installPath As String
    Protected m_dialAttempts As Integer
    Protected m_modem1num As String
    Protected m_modem2num As String
    Protected m_modem3num As String
    Protected m_modem4num As String
    Protected m_isSatAWorkday As Boolean
    Protected m_isSunAWorkday As Boolean
    Protected m_custParams As String
    Protected m_workdayExcdThresh As Integer
    Protected m_otherExcdThresh As Integer
    Protected m_dialInProg As Boolean
    Protected m_saveLogAfterDL As Boolean
    Protected m_modemPorts(3) As String
    Protected m_stopAtNight As Boolean
    Protected m_conn9600 As Boolean
    Protected m_maxLines As Integer


    Protected m_weeknightDL As TimeSpan
    Protected m_weekendDL As TimeSpan
    Protected m_workdayStart As TimeSpan
    Protected m_workdayEnd As TimeSpan

#Region "Properties"
    Public Property InstallPath() As String
        Get
            Return m_installPath
        End Get
        Set(ByVal value As String)
            m_installPath = value
        End Set
    End Property
    Public Property DialAttempts() As Integer
        Get
            Return m_dialAttempts
        End Get
        Set(ByVal value As Integer)
            m_dialAttempts = value
        End Set
    End Property
    Public Property Modem1Num() As String
        Get
            Return m_modem1num
        End Get
        Set(ByVal value As String)
            m_modem1num = value
        End Set
    End Property
    Public Property Modem2Num() As String
        Get
            Return m_modem2num
        End Get
        Set(ByVal value As String)
            m_modem2num = value
        End Set
    End Property
    Public Property Modem3Num() As String
        Get
            Return m_modem3num
        End Get
        Set(ByVal value As String)
            m_modem3num = value
        End Set
    End Property
    Public Property Modem4Num() As String
        Get
            Return m_modem4num
        End Get
        Set(ByVal value As String)
            m_modem4num = value
        End Set
    End Property
    Public Property IsSatAWorkday() As Boolean
        Get
            Return Me.m_isSatAWorkday
        End Get
        Set(ByVal value As Boolean)
            Me.m_isSatAWorkday = value
        End Set
    End Property
    Public Property IsSunAWorkday() As Boolean
        Get
            Return Me.m_isSunAWorkday
        End Get
        Set(ByVal value As Boolean)
            Me.m_isSunAWorkday = value
        End Set
    End Property
    Public Property CustParams() As String
```

```
        Get
            Return m_custParams
        End Get
        Set(ByVal value As String)
            m_custParams = value
        End Set
    End Property
    Public Property WorkdayExcdThresh() As Integer
        Get
            Return m_workdayExcdThresh
        End Get
        Set(ByVal value As Integer)
            Me.m_workdayExcdThresh = value
        End Set
    End Property
    Public Property OtherExcdThresh() As Integer
        Get
            Return Me.m_otherExcdThresh
        End Get
        Set(ByVal value As Integer)
            Me.m_otherExcdThresh = value
        End Set
    End Property
    Public Property DialInProg() As Boolean
        Get
            Return Me.m_dialInProg
        End Get
        Set(ByVal value As Boolean)
            Me.m_dialInProg = value
        End Set
    End Property
    Public Property SaveLogAfterDL() As Boolean
        Get
            Return Me.m_saveLogAfterDL
        End Get
        Set(ByVal value As Boolean)
            Me.m_saveLogAfterDL = value
        End Set
    End Property
    Public Property ModemPorts() As String()
        Get
            Return m_modemPorts
        End Get
        Set(ByVal value() As String)
            m_modemPorts = value
        End Set
    End Property
    Public Property Modem1Port() As String
        Get
            Return m_modemPorts(0)
        End Get
        Set(ByVal value As String)
            m_modemPorts(0) = value
        End Set
    End Property
    Public Property Modem2Port() As String
        Get
            Return m_modemPorts(1)
        End Get
        Set(ByVal value As String)
            m_modemPorts(1) = value
        End Set
    End Property
    Public Property Modem3Port() As String
        Get
            Return m_modemPorts(2)
        End Get
        Set(ByVal value As String)
            m_modemPorts(2) = value
        End Set
    End Property
    Public Property Modem4Port() As String
        Get
            Return m_modemPorts(3)
        End Get
        Set(ByVal value As String)
            m_modemPorts(3) = value
        End Set
    End Property
    Public Property StopAtNight() As Boolean
        Get
            Return m_stopAtNight
        End Get
        Set(ByVal value As Boolean)
            m_stopAtNight = value
        End Set
    End Property
    Public Property Conn9600() As Boolean
        Get
            Return m_conn9600
        End Get
        Set(ByVal value As Boolean)
```

```
            m_conn9600 = value
        End Set
    End Property
    Public Property MaxLines() As Integer
        Get
            Return m_maxLines
        End Get
        Set(ByVal value As Integer)
            If value > 19 Then
                m_maxLines = value
            Else
                m_maxLines = 1000
            End If
        End Set
    End Property


    Public Property WeeknightDL() As TimeSpan
        Get
            Return m_weeknightDL
        End Get
        Set(ByVal value As TimeSpan)
            m_weeknightDL = value
        End Set
    End Property
    Public Property WeekendDL() As TimeSpan
        Get
            Return m_weekendDL
        End Get
        Set(ByVal value As TimeSpan)
            m_weekendDL = value
        End Set
    End Property
    Public Property WorkdayStart() As TimeSpan
        Get
            Return m_workdayStart
        End Get
        Set(ByVal value As TimeSpan)
            m_workdayStart = value
        End Set
    End Property
    Public Property WorkdayEnd() As TimeSpan
        Get
            Return m_workdayEnd
        End Get
        Set(ByVal value As TimeSpan)
            m_workdayEnd = value
        End Set
    End Property
#End Region


    Public Sub New()
        With Me
            .InstallPath = "C:\Program Files\L-D Download\"
            .DialAttempts = 4
            .Modem1Num = "14102721390"
            .Modem2Num = "14102721390"
            .Modem3Num = "14102721390"
            .Modem4Num = "14102721390"
            .Modem1Port = "COM10"
            .Modem2Port = "COM11"
            .Modem3Port = "COM12"
            .Modem4Port = "COM13"
            .IsSatAWorkday = True
            .IsSunAWorkday = False
            .StopAtNight = True
            .CustParams = ""
            .WorkdayExcdThresh = 110
            .OtherExcdThresh = 110
            .DialInProg = False
            .SaveLogAfterDL = True
            .Conn9600 = True
            .MaxLines = 1000
            .WeeknightDL = New TimeSpan(4, 30, 0)
            .WeekendDL = New TimeSpan(4, 30, 0)
            .WorkdayStart = New TimeSpan(6, 0, 0)
            .WorkdayEnd = New TimeSpan(22, 0, 0)
        End With
    End Sub

    Public Sub Serialize()

        Using fs As New FileStream(Me.InstallPath & "app.dat", FileMode.Create)

            Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
            bf.Serialize(fs, Me)
        End Using

    End Sub

End Class
```

```
'Modem
'Instantiate one for each modem attached to a system

'Manages the transfer of data to and from each unit


Imports System
Imports System.IO.Ports
Imports System.Text
Imports System.Text.RegularExpressions
Imports System.Collections.Generic
Imports System.Threading
Imports System.ComponentModel


Public Class Modem

    Protected WithEvents comx As New SerialPort
    Protected m_modemMode As Mode
    Public modemPhoneNum As String = ""
    Public speakerOn As Boolean
    Private getUnitTimeCounter As Integer = 0

    Public Event UpdateOutBoxesEvent As EventHandler(Of UpdateOutBoxesEventArgs)
    'Public Event UpdateStatusBar As EventHandler(Of StatusBarEventArgs)
    Public Event SBUnitCounter As EventHandler(Of StatusBarEventArgs)
    Public Event UpdateMessages As EventHandler(Of StatusBarEventArgs)


    Public Property modemMode() As Mode
        Get
            Return m_modemMode
        End Get
        Set(ByVal value As Mode)
            m_modemMode = value
        End Set
    End Property

    Public Sub New(ByVal portName As String, ByVal phoneNum As String)
        With comx
            .PortName = portName
            .BaudRate = 115200
            .Parity = IO.Ports.Parity.None
            .DataBits = 8
            .StopBits = IO.Ports.StopBits.One
            .Encoding = System.Text.Encoding.UTF8
            .DtrEnable = True   'needed to make MultiModem work
            .RtsEnable = True   'needed to make MultiModem receive text
        End With
        modemPhoneNum = phoneNum
    End Sub

    Public Sub New(ByVal portName As String, ByVal phoneNum As String, ByVal classMode As Mode)
        With comx
            .PortName = portName
            .BaudRate = 115200
            .Parity = IO.Ports.Parity.None
            .DataBits = 8
            .StopBits = IO.Ports.StopBits.One
            .Encoding = System.Text.Encoding.UTF8
            .DtrEnable = True   'needed to make MultiModem work
            .RtsEnable = True   'needed to make MultiModem receive text
        End With
        modemPhoneNum = phoneNum
        m_modemMode = classMode
    End Sub

    Public Enum Mode
        Listen
        Dial
        Standby
        ListenCancel
    End Enum

#Region "Read/Write Engine"

    Public Function ReadLine(ByVal searchChar As Char) As String
        'reads until the search char is found
        'faster than ReadUntilChar
        'also returns nothing if the search character is not found

        Dim returnstr As String = ""
        Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
            New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)
        Dim newLineCharProperty As String = comx.NewLine
        Dim timeOutProperty As Integer = comx.ReadTimeout
        Dim supressDisplay As Boolean = False
        Dim iii As Integer = 1

        comx.NewLine = searchChar
        comx.ReadTimeout = 500

        Do
            Try
                returnstr = comx.ReadLine()
            Catch ex As Exception
                'returnstr = ""
            End Try
            iii += 1
        Loop Until returnstr.Length > 0 OrElse iii > 16 '8 secs

        If returnstr.Length <= 0 Then 'no response received
            returnstr = "No response." & vbLf
            supressDisplay = True
        End If

        'comx.NewLine = searchChar
        'comx.ReadTimeout = 5000

        'Try
        '    returnstr = comx.ReadLine()
        'Catch ex As TimeoutException
        '    returnstr = "No response." & vbLf
        '    supressDisplay = True
        'End Try

        comx.NewLine = newLineCharProperty
        comx.ReadTimeout = timeOutProperty

        If Not supressDisplay Then
            EA.AppendThisString = returnstr '& vbCr
            RaiseEvent UpdateOutBoxesEvent(Me, EA)
        End If

        Return returnstr
    End Function

    Public Function ReadLine(ByVal searchChar As Char, ByVal timeout As Double) As String
        'reads until the search char is found, allows for a custom timeout value
        'faster than ReadUntilChar, but it returns early if no bytes were immediately available to read
        'also returns nothing if the search character is not found

        Dim returnstr As String = ""
        Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
            New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)
        Dim newLineCharProperty As String = comx.NewLine
        Dim timeOutProperty As Integer = comx.ReadTimeout
        Dim supressDisplay As Boolean = False
        Dim iii As Integer = 1

        comx.NewLine = searchChar
        comx.ReadTimeout = 500

        Do
            Try
                returnstr = comx.ReadLine()
            Catch ex As Exception
                'returnstr = ""
            End Try
            iii += 1
        Loop Until returnstr.Length > 0 OrElse iii > CInt(timeout * 2)

        If returnstr.Length <= 0 Then 'no response received
            returnstr = "No response." & vbLf
            supressDisplay = True
        End If

        'comx.NewLine = searchChar
        'comx.ReadTimeout = CInt(timeout * 1000)

        'Try
        '    returnstr = comx.ReadLine()
        'Catch ex As TimeoutException
        '    returnstr = "No response." & vbLf
        '    supressDisplay = True
        'End Try

        comx.NewLine = newLineCharProperty
        comx.ReadTimeout = timeOutProperty

        If Not supressDisplay Then
            EA.AppendThisString = returnstr '& vbCr
            RaiseEvent UpdateOutBoxesEvent(Me, EA)
        End If

        Return returnstr
    End Function

    Public Function ReadLine(ByVal searchChar As Char, ByVal timeout As Integer, _
    ByRef worker As BackgroundWorker, ByRef e As DoWorkEventArgs) As String
        'reads until the search char is found, allows for a custom timeout value
        'exits if a cancel is requested - use this method when waiting for a modem to respond

        Dim count As Integer = 0
        Dim returnstr As String = ""
        Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
            New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)
        Dim newLineCharProperty As String = comx.NewLine
        Dim timeOutProperty As Integer = comx.ReadTimeout

        comx.NewLine = searchChar
        comx.ReadTimeout = 1000

        Do
            Try
                returnstr = comx.ReadLine()
                Exit Do     'if this statement is reached, there is data to return
            Catch ex As TimeoutException
                count += 1
                If worker.CancellationPending Then
                    Throw New CancelException
                End If
            End Try
        Loop Until count >= timeout

        comx.NewLine = newLineCharProperty
        comx.ReadTimeout = timeOutProperty

        EA.AppendThisString = returnstr '& vbCr
        RaiseEvent UpdateOutBoxesEvent(Me, EA)

        Return returnstr

    End Function

    Public Sub Write(ByVal value As String)
        'Writes the passed text to the serial port and also to the appropriate window
        Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
            New UpdateOutBoxesEventArgs(value, UpdateOutBoxesEventArgs.RW.Write)

        comx.Write(value)
        RaiseEvent UpdateOutBoxesEvent(Me, EA)

    End Sub

    Public Sub Notify(ByVal value As String)
        Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
                    New UpdateOutBoxesEventArgs(value, UpdateOutBoxesEventArgs.RW.Notify)

        RaiseEvent UpdateOutBoxesEvent(Me, EA)
    End Sub

#End Region

    Public Delegate Function GetUnitDelegate(ByVal lnum As String) As LDUnit
    Public Delegate Sub ReturnUnitDelegate(ByVal unit As LDUnit)

    Public Sub Listen(ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs)
        'listen on one modem for incoming calls
        'will run on a separate thread, analogous to Dial

        If modemMode = Mode.Dial OrElse worker.CancellationPending Then
            Exit Sub
        End If

        Dim unit As New LDUnit("temp")  'Temporary unit object
        Dim result As String = ""
        Dim result2 As String = ""
        Dim uNum As String = ""
        Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab)}
        Dim deleg As New GetUnitDelegate(AddressOf Form1.GetUnit)
        Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)
        Dim newLineChar As String = comx.NewLine
        Dim ReadTimeOutPropertyValue As Integer = comx.ReadTimeout
        Dim lockcode As String = "22222222"
        Dim serial As String = ""

        Try
            If comx.IsOpen Then
                Notify("Port " & comx.PortName & " is in use by another process" & vbLf)
                Exit Sub
                'comx.Close()
            End If
            comx.Open()
            comx.DiscardInBuffer()  'flush receive buffer

            Notify("*** Monitoring for incoming calls ***" & vbLf & vbLf)
            Write("AT" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            comx.DiscardInBuffer()
            Write("ATZ" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            'MessageBox.Show(result, "result after first write")
            If Not result Like "*OK*" Then
```

```
                comx.DiscardInBuffer()
                Write("+++")
                If comx.PortName <> "COM1" Then
                    ReadLine(CChar(vbLf), 0.5)
                    ReadLine(CChar(vbLf), 0.5)
                End If
                Write("ATH" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
                Thread.Sleep(1500)

                comx.DiscardInBuffer()
                Write("AT" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
                'MessageBox.Show(result, "result after second AT command")
                If Not result Like "*OK*" Then
                    Throw New NoModemException("Modem on " & comx.PortName & " not present or responsive")
                End If
            End If

            If speakerOn Then
                Write("ATM1" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
            End If

            If Form1.appset.Conn9600 Then
                Write("AT+MS=V32,1,2400,9600,2400,9600" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
            End If

            'wait forever until a "Ring" is received
            comx.NewLine = vbLf
            comx.ReadTimeout = 1000
            Dim speakerold As Boolean = speakerOn

            Do
                Try
                    If worker.CancellationPending Then
                        Throw New CancelException
                    End If
                    If speakerold <> speakerOn Then
                        If speakerOn Then
                            Write("ATM1" & vbCr)
                            result = ReadLine(CChar(vbLf))
                            result &= ReadLine(CChar(vbLf))
                        Else
                            Write("ATM0" & vbCr)
                            result = ReadLine(CChar(vbLf))
                            result &= ReadLine(CChar(vbLf))
                        End If
                        speakerold = speakerOn
                    End If
                    result = comx.ReadLine
                Catch ex As TimeoutException
                    'no more data to read
                End Try
            Loop Until result Like "*RING*"

            comx.ReadTimeout = ReadTimeOutPropertyValue
            comx.NewLine = newLineChar

            Write("ATA" & vbCr)     'Answer call
            result = ReadLine(CChar(vbLf), 90, worker, e)
            result &= ReadLine(CChar(vbLf), 90, worker, e)
            If worker.CancellationPending Then
                Throw New CancelException
            End If
            If result Like "*CONNECT*" Then     'get password and unit number
                unit.LockCode = lockcode      'only APG units call in
                unit = getPass(unit, worker, e)
                serial = unit.UnitSerial

                'get unit number and ask for the appropriate LDUnit object
                Write("Q157" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
                If result Like "*ARNING*" Then
                    Write("Q157" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim(trimChars)
                End If
                If result Like "*No response.*" Then  'ReadLine timed out
                    comx.DiscardInBuffer()
                    Write("Q157" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim(trimChars)
                    result2 = ReadLine(CChar(vbLf), 1)
                    result2 = result2.Trim(trimChars)
                    If Not result Like "*No response.*" Then       'use first read entry
```

```
                        uNum = result
                    ElseIf Not result2 Like "*No response.*" Then    'use second read entry
                        uNum = result2
                    Else
                        Throw New FormatException("Can't read from unit - will try again later")
                    End If
                Else
                    uNum = result
                End If
                If uNum.Length < 2 Then   'This operation assigns an L## LDunit to the unit variable
                    unit = deleg.Invoke("L0" & uNum)
                Else
                    unit = deleg.Invoke("L" & uNum)
                End If

                If unit.UnitNum = "null" Then
                    Throw New Exception("Unit is being accessed by another part of the program." & vbLf)
                End If

                unit.UnitSerial = serial
                'unit.ResetDataYN = True
                Download(unit, False, worker, e)

                unit.DialTries = 0
                unit.DLTries = 0
                Write("M12" & vbCr)
                result = ReadLine(CChar(vbLf), 10)
                comx.DiscardInBuffer()
                Write("+++")
                If comx.PortName <> "COM1" Then
                    ReadLine(CChar(vbLf), 0.5)
                    ReadLine(CChar(vbLf), 0.5)
                End If
                Write("ATH" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))

                'comx.Close() 'moved to Finally block

            Else        'call was dropped
                Throw New NoConnectException("Unable to connect to remote modem")
            End If

        Catch ex As NoModemException    'don't use this modem to download the remaining units
            Notify("Error: " & ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            If worker.CancellationPending Then
                e.Cancel = True
            End If
            If comx.IsOpen Then
                comx.Close()
            End If
            Exit Sub
        Catch ex As NoConnectException      'what happens when the line is busy or unavailable
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
            Notify("Error: " & ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            unit.DialTries += 1
            If unit.DialTries >= Form1.appset.DialAttempts Then
                'stop trying to download unit
                unit.IsDownloadDone = True
                unit.DialTries = 0
                Dim DA As Integer = Form1.appset.DialAttempts + 1
                Notify("Attempted to dial unit " & unit.UnitNum & " " & DA.ToString & " times." & vbLf)
                Notify("Will not try again until the next Dial command is issued." & vbLf)
            End If
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Catch ex As CancelException     'user cancels the operation - close the port and exit
            Notify(ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            e.Cancel = True
            If comx.IsOpen Then
                comx.Close()
            End If
            Exit Sub
        Catch ex As InvalidOperationException      'port is closed when attempting to write data
            Notify("InvalidOperationException" & vbLf)
            Notify("Error: " & ex.Message & vbLf)
            'If comx.IsOpen Then
```

```vb
            '    comx.Close()
            'End If
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Catch ex As IO.IOException      'port is closed when attempting to write data
            Notify("IOException" & vbLf)
            Notify("Error: " & ex.Message & vbLf)
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Catch ex As Exception
            'will usually be from the download routine, and result from a dropped
            'or error-prone connection

            Write("M12" & vbCr)     'tell 870 to shut down
            result = ReadLine(CChar(vbLf), 10)
            'result &= ReadLine(CChar(vbLf), 10)
            comx.DiscardInBuffer()
            Write("+++")
            If comx.PortName <> "COM1" Then
                ReadLine(CChar(vbLf), 0.5)
                ReadLine(CChar(vbLf), 0.5)
            End If
            Write("ATH" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
            Notify("Error: " & ex.Message & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If

        Finally
            If comx.IsOpen Then
                comx.Close()
            End If
        End Try

        If worker.CancellationPending Then
            e.Cancel = True
            Exit Sub
        End If

        'conserve memory while the listen routine is recursing
        ' - the "unit" has been sent back to LDUnitList
        unit.EList.Clear()
        unit.IList.Clear()
        unit.RList.Clear()
        unit.CList.Clear()
        unit.QList.Clear()
        unit.LList.Clear()

        'listen for the next unit to call
        Listen(worker, e)
    End Sub

    Public Sub Dial(ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs)
        'dials a remote modem

        If modemMode = Mode.Listen OrElse worker.CancellationPending Then
            Exit Sub
        End If

        Dim result As String = ""
        Dim deleg As New GetUnitDelegate(AddressOf Form1.GetUnit)
        Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)
        Dim unit As LDUnit = deleg.Invoke("-2")


        If String.Compare(unit.UnitNum, "null") = 0 Then
            Form1.appset.DialInProg = False 'dial finished, stop dialing units
            Exit Sub
        End If

        Notify("*** Now dialing unit " & unit.UnitNum & " at " & Date.Now.ToString("M/d/yyyy HH:mm:ss") _
        & " ***" & vbLf)

        Try
            If unit.IsEnabled = False Then
                Throw New NotEnabledException
            End If

            If comx.IsOpen Then
                Notify("Port " & comx.PortName & " is in use by another process" & vbLf)
                deleg2.Invoke(unit)
                Exit Sub
                'comx.Close()
            End If
            comx.Open()
            comx.DiscardInBuffer()  'flush receive buffer

            Write("AT" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            comx.DiscardInBuffer()
            Write("ATZ" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            'MessageBox.Show(result, "result after first write")
            If Not result Like "*OK*" Then
                comx.DiscardInBuffer()
                Write("+++")
                If comx.PortName <> "COM1" Then
                    ReadLine(CChar(vbLf), 0.5)
                    ReadLine(CChar(vbLf), 0.5)
                End If
                Write("ATH" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
                Thread.Sleep(1500)

                comx.DiscardInBuffer()
                Write("AT" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
                'MessageBox.Show(result, "result after second AT command")
                If Not result Like "*OK*" Then
                    Throw New NoModemException("Modem on " & comx.PortName & " not present or responsive")
                End If
            End If
            If speakerOn Then
                Write("ATM1" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
            End If
            If Form1.appset.Conn9600 Then
                Write("AT+MS=V32,1,2400,9600,2400,9600" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
            End If

            'If unit.UnitOwner = LDUnit.Owner.CERL Then  'limit modem speed to 14400 bps
            '    If comx.PortName = "COM1" Then
            '        Write("AT$MB14400 \N3" & vbCr)
            '        result = ReadLine(CChar(vbLf))
            '        result &= ReadLine(CChar(vbLf))
            '    Else
            '        Write("AT+MS=V32B,1,300,14400,300,14400" & vbCr)
            '        result = ReadLine(CChar(vbLf))
            '        result &= ReadLine(CChar(vbLf))
            '        Write("AT\N3" & vbCr)
            '        result = ReadLine(CChar(vbLf))
            '        result &= ReadLine(CChar(vbLf))
            '    End If
            'End If

            Write("ATDT" & unit.UnitPhoneNum & vbCr)
            result = ReadLine(CChar(vbLf), 90, worker, e)
            result &= ReadLine(CChar(vbLf), 90, worker, e)
            If worker.CancellationPending Then
                Throw New CancelException
            End If
            If result Like "*OK*" Then
                result &= ReadLine(CChar(vbLf), 90, worker, e)
                If worker.CancellationPending Then
                    Throw New CancelException
                End If
            End If
            If Not result Like "*CONNECT*" Then
                comx.DiscardInBuffer()
                Write("ATZ" & vbCr)
                result = ReadLine(CChar(vbLf))
                result &= ReadLine(CChar(vbLf))
                If speakerOn Then
                    Write("ATM1" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result &= ReadLine(CChar(vbLf))
                End If
                If Form1.appset.Conn9600 Then
                    Write("AT+MS=V32,1,2400,9600,2400,9600" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result &= ReadLine(CChar(vbLf))
                End If

                Write("ATDT" & unit.UnitPhoneNum & vbCr)
                result = ReadLine(CChar(vbLf), 90, worker, e)
                result &= ReadLine(CChar(vbLf), 90, worker, e)
                If worker.CancellationPending Then
                    Throw New CancelException
                End If
                If result Like "*OK*" Then
                    result &= ReadLine(CChar(vbLf), 90, worker, e)
                    If worker.CancellationPending Then
                        Throw New CancelException
                    End If
                End If
                If Not result Like "*CONNECT*" Then
                    Throw New NoConnectException("Unable to connect to remote modem")
                End If
            End If

            If unit.Initialize Then
                ReconfigureUnit(unit, True, worker, e)
            Else
                Download(unit, True, worker, e)  'executes download on same thread
            End If

            comx.DiscardInBuffer()
            Write("M12" & vbCr)
            result = ReadLine(CChar(vbLf), 10)
            'result &= ReadLine(CChar(vbLf), 10)
            comx.DiscardInBuffer()
            Write("+++")
            If comx.PortName <> "COM1" Then
                ReadLine(CChar(vbLf), 0.5)
                ReadLine(CChar(vbLf), 0.5)
            End If
            Write("ATH" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))


            'comx.Close() ' moved to Finally block

        Catch ex As NoModemException    'don't use this modem to download the remaining units
            Notify("Error: " & ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            If worker.CancellationPending Then
                e.Cancel = True
            End If
            If comx.IsOpen Then
                comx.Close()
            End If
            Exit Sub
        Catch ex As NoConnectException      'what happens when the line is busy or unavailable
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
            Notify("Error: " & ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            unit.DialTries += 1
            If unit.DialTries >= Form1.appset.DialAttempts Then
                'stop trying to download unit
                unit.IsDownloadDone = True
                unit.DialTries = 0
                Dim DA As Integer = Form1.appset.DialAttempts + 1
                Notify("Attempted to dial unit " & unit.UnitNum & " " & DA.ToString & " times." & vbLf)
                Notify("Will not try again until the next Dial command is issued." & vbLf)
            End If
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Catch ex As NotEnabledException
            Notify("Skipping unit " & unit.UnitNum & " because is not enabled." & vbLf)
            Dim EA As New StatusBarEventArgs(unit.UnitNum, StatusBarEventArgs.SBAction.Remove)
            'RaiseEvent UpdateStatusBar(Me, EA)
            RaiseEvent SBUnitCounter(Me, EA)
            RaiseEvent UpdateMessages(Me, EA)

            'If comx.IsOpen Then
            '    comx.Close()
            'End If
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = True
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Catch ex As CancelException     'user cancels the operation - close the port and exit
            Notify(ex.msg & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            e.Cancel = True
            If comx.IsOpen Then
                comx.Close()
            End If
            Exit Sub
        Catch ex As InvalidOperationException      'port is closed when attempting to write data
            Notify("InvalidOperationException" & vbLf)
            Notify("Error: " & ex.Message & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
        Catch ex As IO.IOException      'port is closed when attempting to write data
            Notify("IOException" & vbLf)
            Notify("Error: " & ex.Message & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
            'If comx.IsOpen Then
            '    comx.Close()
            'End If
        Catch ex As Exception
            'will usually be from the download routine, and result from a dropped
            'or error-prone connection

            Write("M12" & vbCr)     'tell 870 to shut down
            result = ReadLine(CChar(vbLf), 10)
            'result &= ReadLine(CChar(vbLf), 10)
            Write("+++")
            If comx.PortName <> "COM1" Then
                ReadLine(CChar(vbLf), 0.5)
                ReadLine(CChar(vbLf), 0.5)
            End If
            Write("ATH" & vbCr)
            result = ReadLine(CChar(vbLf))
            result &= ReadLine(CChar(vbLf))
            Notify("Error: " & ex.Message & vbLf)
            unit.IsDownloadInProgress = False
            unit.IsDownloadDone = False
            unit.DialTries += 1
            If unit.DialTries > Form1.appset.DialAttempts Then
                'stop trying to download unit
                unit.IsDownloadDone = True
                Dim DA As Integer = Form1.appset.DialAttempts + 1
                Notify("Attempted to dial unit " & unit.UnitNum & " " & DA.ToString & " times." & vbLf)
                Notify("Will not try again until the next Dial command is issued." & vbLf)
            End If
            If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
                deleg2.Invoke(unit)
            End If
        Finally
            If comx.IsOpen Then
                comx.Close()
            End If
        End Try

        If worker.CancellationPending Then
            e.Cancel = True
            Exit Sub
        End If

        'conserve memory while the listen routine is recursing
        ' - the "unit" has been sent back to LDUnitList
        unit.EList.Clear()
        unit.IList.Clear()
        unit.RList.Clear()
        unit.CList.Clear()
        unit.QList.Clear()
        unit.LList.Clear()

        'repeat until all units are done
        Dial(worker, e)
    End Sub

    Public Sub td(ByVal unit As LDUnit, ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs)
        'perform some routine on all units
        Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)

        Dim result As String = ""
        Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab)}

        unit = getPass(unit, worker, e)

        Write("R93" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*@*" Then       'turn off L-D error checking
            Write("S205,0F" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("R93" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        ElseIf result Like "*ARNING*" Then
            Write("R93" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If

        If worker.CancellationPending Then
            e.Cancel = True
            Throw New CancelException
        End If

        Write("Q158" & vbCr)
        result = ReadLine(CChar(vbLf))

        If worker.CancellationPending Then
            Write("M12" & vbCr)
            Throw New CancelException
        End If

        Write("S158,X4 E0 Q0 V0 T M1 S0=5 &D3" & vbCr)
        result = ReadLine(CChar(vbLf))
        Write("Q158" & vbCr)
        result = ReadLine(CChar(vbLf))

    End Sub

    Public Sub Download(ByVal unit As LDUnit, ByVal askPass As Boolean, _
    ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs)
        'Initiates a data transfer from the 870
        'This procedure is common to both the dial out and dial in routines
        'This procedure assumes the host computer is connected to the 870

        Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)

        Dim result As String = ""
        Dim result2 As String = ""
        Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab)}
        Dim parseRes As Boolean = True
        Dim parseRes2 As Boolean = True
        Dim excdThreshTemp As Double = -1
        Dim calLevel As Double = -1

        Dim numExceedances As Integer = -1   'r93, E records
        Dim numIntervals As Integer = -1     'r94, I records
        Dim numStartStops As Integer = -1    'r92, L records
        Dim numCalibrations As Integer = -1  'r97, C records
        Dim numreportlines As Integer = 0
        Dim battVoltage As Double = -1.1     'r86
        Dim unitErrorString As String = ""   'r98
        Dim lotsOfExcds As Boolean = False
        Dim lotsOfIntvs As Boolean = False
        Dim report As List(Of String) = New List(Of String)     'the report, each line is a List entry
        Dim EA2 As UpdateOutBoxesEventArgs = _
                    New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)


        unit.IsDownloadDone = False
        unit.IsDownloadInProgress = False
        If Not comx.IsOpen Then
            deleg2.Invoke(unit)
            Exit Sub
        End If

        unit.IsDownloadInProgress = True

        'reset report data
        unit.NumCalibrations = -1
        unit.NumExceedances = -1
        unit.NumIntervals = -1
        unit.NumStartStops = -1
        unit.ErrorString = ""
        unit.BattVoltage = -1
        unit.EList.Clear()
        unit.IList.Clear()
        unit.RList.Clear()
        unit.QList.Clear()
        unit.LList.Clear()
        unit.CList.Clear()

        'retrieve password, cancel download if it is in progress
        If askPass Then
            unit = getPass(unit, worker, e)
        End If

        'retrieve device status
        'to put read data into local variables, trim the cr, lf, : and spaces from the string,
        'then parse it to an integer or double

        'RemoveHandler comx.DataReceived, AddressOf ReadPort
        Notify("*** Downloading unit " & unit.UnitNum & " at " & Date.Now.ToString("M/d/yyyy HH:mm:ss") _
        & " ***" & vbLf)

        Write("R93" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*@*" Then       'turn off L-D error checking
            Write("S205,0F" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("R93" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        ElseIf result Like "*ARNING*" Then
            Write("R93" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempint, tempint2 As Integer

            comx.DiscardInBuffer()
            Write("R93" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            parseRes = Integer.TryParse(result, tempint)
            parseRes2 = Integer.TryParse(result2, tempint2)
            If parseRes2 Then       'use second read entry
                numExceedances = tempint2
            ElseIf parseRes Then    'use first read entry
                numExceedances = tempint
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            numExceedances = Integer.Parse(result)
        End If

        If numExceedances > 50 Then
            lotsOfExcds = True
        End If


        Write("R94" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*ARNING*" Then
            Write("R94" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempint, tempint2 As Integer

            comx.DiscardInBuffer()
            Write("R94" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            parseRes = Integer.TryParse(result, tempint)
            parseRes2 = Integer.TryParse(result2, tempint2)
            If parseRes2 Then       'use second read entry
                numIntervals = tempint2
            ElseIf parseRes Then    'use first read entry
                numIntervals = tempint
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            numIntervals = Integer.Parse(result)
        End If

        If numIntervals > 50 Then
            lotsofintvs = True
        End If


        Write("R92" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*ARNING*" Then
```

```
            Write("R92" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempint, tempint2 As Integer

            comx.DiscardInBuffer()
            Write("R92" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            parseRes = Integer.TryParse(result, tempint)
            parseRes2 = Integer.TryParse(result2, tempint2)
            If parseRes2 Then       'use second read entry
                numStartStops = tempint2
            ElseIf parseRes Then    'use first read entry
                numStartStops = tempint
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            numStartStops = Integer.Parse(result)
        End If


        Write("R97" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*ARNING*" Then
            Write("R97" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempint, tempint2 As Integer

            comx.DiscardInBuffer()
            Write("R97" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            parseRes = Integer.TryParse(result, tempint)
            parseRes2 = Integer.TryParse(result2, tempint2)
            If parseRes2 Then       'use second read entry
                numCalibrations = tempint2
            ElseIf parseRes Then    'use first read entry
                numCalibrations = tempint
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            numCalibrations = Integer.Parse(result)
        End If


        Dim trimchars2() As Char = {"V"c}
        Write("R86" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        result = result.Trim(trimchars2)
        If result Like "*ARNING*" Then
            Write("R86" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result = result.Trim(trimchars2)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempd, tempd2 As Double

            comx.DiscardInBuffer()
            Write("R86" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result = result.Trim(trimchars2)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            result2 = result2.Trim(trimchars2)
            parseRes = Double.TryParse(result, tempd)
            parseRes2 = Double.TryParse(result2, tempd2)
            If parseRes2 Then       'use second read entry
                battVoltage = tempd2
            ElseIf parseRes Then    'use first read entry
                battVoltage = tempd
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            battVoltage = Double.Parse(result)
        End If
        If battVoltage > 20 Then
            battVoltage *= 0.1
        End If

        Write("R98" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*ARNING*" Then
            Write("R98" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            comx.DiscardInBuffer()
            Write("R98" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            If Not result Like "*No response.*" Then       'use first read entry
                unitErrorString = result
            ElseIf Not result2 Like "*No response.*" Then    'use second read entry
                unitErrorString = result2
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            unitErrorString = result
        End If

        'subtract 128 from each warning code to make codes easier to read
        'Actual errors (rare) are numbered 1 thru 8 - add 100 to them to differentiate from the
        'more common warning codes
        Dim trimChars3() As Char = {","c}
        Dim esTemp() As String = unitErrorString.Split(trimChars3, StringSplitOptions.RemoveEmptyEntries)
        Dim errorInts(esTemp.Length - 1) As Integer

        unitErrorString = ""

        For i As Integer = 0 To esTemp.Length - 1
            Dim tryp As Boolean = Integer.TryParse(esTemp(i), errorInts(i))
            If tryp Then 'add or subtract and append result to the error string
                If errorInts(i) > 127 Then
                    errorInts(i) -= 128 'warning codes now numbered 1-43
                Else
                    errorInts(i) += 100 'error codes now numbered 101-108
                End If
                If i < esTemp.Length - 1 Then
                    unitErrorString &= errorInts(i).ToString & ","
                Else
                    unitErrorString &= errorInts(i).ToString
                End If
                'End If
            End If
        Next



        'get cal level
        Write("R107" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If result Like "*ARNING*" OrElse result.Length > 7 Then
            Write("R107" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
        End If
        If result Like "*No response.*" Then  'ReadLine timed out
            Dim tempd, tempd2 As Double

            comx.DiscardInBuffer()
            Write("R107" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            result2 = ReadLine(CChar(vbLf), 1)
            result2 = result2.Trim(trimChars)
            parseRes = Double.TryParse(result, tempd)
            parseRes2 = Double.TryParse(result2, tempd2)
            If parseRes2 Then       'use second read entry
                calLevel = tempd2
            ElseIf parseRes Then    'use first read entry
                calLevel = tempd
            Else
                Throw New FormatException("Can't read from unit - will try again later")
            End If
        Else
            calLevel = Double.Parse(result)
        End If


        'MessageBox.Show(numExceedances.ToString & vbCr & numIntervals.ToString & vbCr & numStartStops.  ↙
    ToString & vbCr & result & vbCr & battVoltage)
```

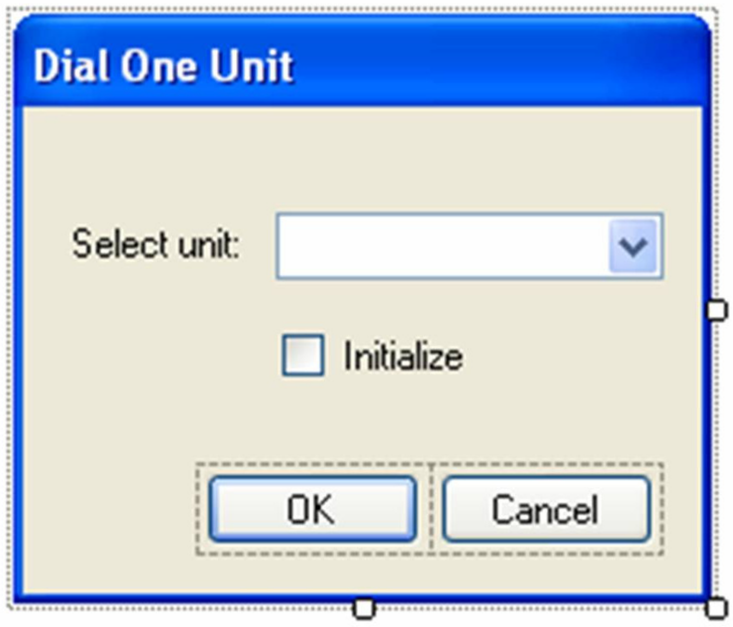
```

        If battVoltage < 0 OrElse numStartStops < 0 OrElse numIntervals < 0 OrElse numExceedances < 0 Then
            unit.IsDownloadInProgress = False
            deleg2.Invoke(unit)
            Exit Sub
        End If



        'make sure all units are in battery cut-off mode, since they run on solar power
        'and will require a battery change if they are left on too long

        Write("Q159" & vbCr)
        result = ReadLine(CChar(vbLf))
        result = result.Trim(trimChars)
        If Not result Like "*Ext*" Then
            Write("S159,1" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S159,1" & vbCr)
            result = ReadLine(CChar(vbLf))
        End If



        'check exceedance threshold
        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
            Write("Q64" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If result Like "*ARNING*" Then
                Write("Q64" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
            End If
            If result Like "*No response.*" Then  'ReadLine timed out
                Dim tempd, tempd2 As Double

                comx.DiscardInBuffer()
                Write("Q64" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
                result2 = ReadLine(CChar(vbLf), 1)
                result2 = result2.Trim(trimChars)
                parseRes = Double.TryParse(result, tempd)
                parseRes2 = Double.TryParse(result2, tempd2)
                If parseRes2 Then       'use second read entry
                    excdThreshTemp = tempd2
                ElseIf parseRes Then    'use first read entry
                    excdThreshTemp = tempd
                Else
                    Throw New FormatException("Can't read from unit - will try again later")
                End If
            Else
                excdThreshTemp = Double.Parse(result)
            End If
        Else
            Write("Q64" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If result Like "*ARNING*" Then
                Write("Q64" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
            End If
            If result Like "*No response.*" Then  'ReadLine timed out
                Dim tempd, tempd2 As Double

                comx.DiscardInBuffer()
                Write("Q64" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
                result2 = ReadLine(CChar(vbLf), 1)
                result2 = result2.Trim(trimChars)
                parseRes = Double.TryParse(result, tempd)
                parseRes2 = Double.TryParse(result2, tempd2)
                If parseRes2 Then       'use second read entry
                    excdThreshTemp = tempd2
                ElseIf parseRes Then    'use first read entry
                    excdThreshTemp = tempd
                Else
                    Throw New FormatException("Can't read from unit - will try again later")
                End If
            Else
                excdThreshTemp = Double.Parse(result)
            End If
        End If



        'Get the unit time and the PC time to check the time difference

        Dim unitDateAvg As Date
        Dim hostDateAvg As Date

        Dim RTcurrent As ReferenceTime = GetUnitTime(worker, e)
        unitDateAvg = RTcurrent.UnitTimeZero
        hostDateAvg = RTcurrent.HostTimeZero

        'MessageBox.Show(unitDateAvg & vbLf & hostDateAvg, "Unit Time, Host Time")

        'Rounding statement - don't convert offset to an integer until it's time to paste the value into the ↙
     report
        'tOffsetAvg1 = CInt(Math.Round(tOffsetAvg1, 0, MidpointRounding.AwayFromZero))

        'for testing only
        'unit.IncludeR = True
        'unit.IncludeQ = True
        'unit.IncludeLC = True
        'for testing only



        'Download the report, per the parameters set in the unit object
        Write("P10" & vbCr)
        result = ReadLine(CChar(vbLf))
        Write("P10" & vbCr)
        result = ReadLine(CChar(vbLf))
        Write("P10" & vbCr)
        result = ReadLine(CChar(vbLf))
        If Not unit.IncludeR Then
            Write("S177,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S177,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not result Like "" Then     'send command again
                Write("S177,0" & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
        Else
            numreportlines += 220
        End If
        If Not unit.IncludeLC Then
            Write("S178,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S178,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not result Like "" Then     'send command again
                Write("S178,0" & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
        Else
            numreportlines += numStartStops
            numreportlines += numCalibrations
        End If
        If Not unit.IncludeQ Then
            Write("S179,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S179,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not result Like "" Then     'send command again
                Write("S179,0" & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
        Else
            numreportlines += 231
        End If

        numreportlines += numExceedances
        numreportlines += numIntervals

        'MessageBox.Show(numReportLines.ToString)

        Dim RQCounter As Integer = 0
        Dim ECounter As Integer = 0
        Dim ICounter As Integer = 0
        Dim newLineChar As String = comx.NewLine
        Dim ReadTimeOutPropertyValue As Integer = comx.ReadTimeout
        comx.NewLine = vbLf
        comx.ReadTimeout = 4000

        Try
            Write("P100" & vbCr)
            For i As Integer = 1 To numreportlines + 2000
                If worker.CancellationPending Then
                    e.Cancel = True
                    Write("M12" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    Throw New CancelException
                End If
                report.Add(comx.ReadLine)
                If (IsR(report.Item(i - 1)) OrElse IsQ(report.Item(i - 1)) _
```

```vb
                OrElse IsL(report.Item(i - 1)) OrElse IsC(report.Item(i - 1))) Then
                    If RQCounter < 80 Then
                        EA2.AppendThisString = "."
                        RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        RQCounter += 1
                    Else
                        EA2.AppendThisString = vbLf & "."
                        RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        RQCounter = 1
                    End If
                End If
                If IsE(report.Item(i - 1)) Then
                    If lotsOfExcds Then  'don't print each excd record to screen
                        ECounter += 1
                        If ECounter = 1 Then
                            EA2.AppendThisString = "There are " & numExceedances.ToString & _
                            " exceedance records on unit." & vbLf & _
                            "Reading Exceedances... "
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        ElseIf ECounter Mod 5 = 0 Then
                            EA2.AppendThisString = "."
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                            If ECounter Mod 50 = 0 Then
                                EA2.AppendThisString = vbLf & ECounter.ToString & " of " & _
                                numExceedances.ToString & " exceedances read"
                                RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                            End If
                        ElseIf ECounter >= numExceedances Then
                            EA2.AppendThisString = vbLf & "All exceedances read." & vbLf
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        End If

                    Else
                        EA2.AppendThisString = report.Item(i - 1) '& vbLf
                        RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                    End If
                End If
                If IsI(report.Item(i - 1)) Then
                    If lotsOfIntvs Then  'don't print each interval record to screen
                        ICounter += 1
                        If ICounter = 1 Then
                            EA2.AppendThisString = "There are " & numIntervals.ToString & _
                            " interval records on unit." & vbLf & _
                            "Reading Intervals... "
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        ElseIf ICounter Mod 5 = 0 Then
                            EA2.AppendThisString = "."
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                            If ICounter Mod 50 = 0 Then
                                EA2.AppendThisString = vbLf & ICounter.ToString & " of " & _
                                numIntervals.ToString & " intervals read"
                                RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                            End If
                        ElseIf ICounter >= numIntervals Then
                            EA2.AppendThisString = vbLf & "All intervals read." & vbLf
                            RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                        End If

                    Else
                        EA2.AppendThisString = report.Item(i - 1) '& vbLf
                        RaiseEvent UpdateOutBoxesEvent(Me, EA2)
                    End If
                End If
            Next
        Catch ex As TimeoutException
            'no more data to read
        End Try

        comx.ReadTimeout = ReadTimeOutPropertyValue
        comx.NewLine = newLineChar

        'MessageBox.Show(unitDateAvg.ToString & vbCr & hostDateAvg.ToString)



        'Check format of the report, if all is well proceed, otherwise download the report again

        Dim EList As New List(Of String)
        Dim IList As New List(Of String)
        Dim LList As New List(Of String)
        Dim CList As New List(Of String)
        Dim RList As New List(Of String)
        Dim QList As New List(Of String)
        Dim downloadAgain As Boolean = False
        Dim IsBadRecords As Boolean = False

        'MessageBox.Show(report.Item(0) & vbCr & report.Item(0).Length.ToString)

        If report.Count < numreportlines Then       'repeat download immediately if DL is clearly not ↙
    complete
            Write("P999" & vbCr)
            Thread.Sleep(1500)
            comx.DiscardInBuffer()
            Write("P999" & vbCr)
            Thread.Sleep(1500)
            comx.DiscardInBuffer()
            unit.DLTries += 1
            If unit.DLTries < 4 Then
                Notify("Report is incomplete - downloading again..." & vbLf)
                Download(unit, False, worker, e)
                Exit Sub
            Else
                unit.IsDownloadInProgress = False
                unit.IsDownloadDone = False
                unit.DLTries = 0
                Notify("Warning! Unit " & unit.UnitNum & " has a bad connection" & vbLf)
                Notify("   and cannot transfer its data." & vbLf)
                Notify("Download will resume after the other units have finished." & vbLf)
                deleg2.Invoke(unit)
                Exit Sub
            End If
        End If

        'Copy valid entries to their respective list
        EList = report.FindAll(AddressOf IsE)
        IList = report.FindAll(AddressOf IsI)

        If EList.Count < numExceedances OrElse IList.Count < numIntervals Then
            IsBadRecords = True
        End If

        If unit.IncludeR Then
            RList = report.FindAll(AddressOf IsR)
            If RList.Count < 220 Then
                IsBadRecords = True
            End If
        End If
        If unit.IncludeQ Then
            QList = report.FindAll(AddressOf IsQ)
            If QList.Count < 231 Then
                IsBadRecords = True
            End If
        End If
        If unit.IncludeLC Then
            LList = report.FindAll(AddressOf IsL)
            CList = report.FindAll(AddressOf IsC)
            If LList.Count < numStartStops OrElse CList.Count < numCalibrations Then
                IsBadRecords = True
            End If
        End If

        'MessageBox.Show(EList.Count.ToString & vbCr & IList.Count.ToString & vbCr & RList.Count & vbCr & ↙
    QList.Count & vbCr & LList.Count & vbCr & CList.Count, "E, I, R, Q, L, C")

        If IsBadRecords Then
            'find and display bad or corrupted Report entries
            Dim badList As List(Of String) = report.FindAll(AddressOf IsBadEntry)
            Dim badListStr As String = ""

            For i As Integer = 1 To badList.Count
                badListStr &= badList.Item(i - 1) & vbCrLf
            Next

            Using sw As New IO.StreamWriter(unit.InstallPath & "Bad Records\error" & unit.UnitNum & _
            " - " & hostDateAvg.ToLongDateString & ".txt", True)
                sw.WriteLine(hostDateAvg.ToString)
                sw.WriteLine()
                sw.Write(badListStr)
                'sw.WriteLine()
                sw.WriteLine("---------------------------------------------------------------")
                sw.WriteLine()
            End Using

            Notify("Warning:" & vbLf)
            Notify(vbLf)
            Notify("The following records from unit " & unit.UnitNum & " are invalid:" & vbLf)
            Notify(vbLf)
            Notify(badListStr & vbLf & vbLf)
            Notify("These records have been written to the file " & unit.InstallPath & _
            "Bad Records\error" & unit.UnitNum & " - " & _
            hostDateAvg.ToLongDateString & ".txt" & vbLf & vbLf)
            Notify("Trying download again..." & vbLf)

            downloadAgain = True
        End If

        If downloadAgain Then
            If unit.DLTries < 4 Then
                'try to cancel the current download
                Write("P999" & vbCr)
                Thread.Sleep(1000)
                Write("P999" & vbCr)
                Thread.Sleep(1000)
                comx.DiscardInBuffer()
```

```vb
                'check to see if unit is still connected
                Write("R1" & vbCr)
                result = ReadLine(CChar(vbLf))
                If result Like "*Larson*" Then
                    unit.DLTries += 1
                    Download(unit, False, worker, e)
                    Exit Sub
                Else
                    unit.IsDownloadInProgress = False
                    unit.IsDownloadDone = False
                    Notify("Warning! Unit " & unit.UnitNum & " has timed out." & vbCr)
                    Notify("Download will resume after the other units have finished." & vbCr)

                    'return unit to the LDUnitList,
                    '(it will be downloaded at a later point in time, after other units have been downloaded)
                    deleg2.Invoke(unit)
                    Exit Sub
                End If
            Else
                unit.IsDownloadInProgress = False
                unit.IsDownloadDone = False
                unit.DLTries = 0
                Notify("Warning! Unit " & unit.UnitNum & " has a bad connection" & vbLf)
                Notify("   and cannot transfer its data." & vbLf)
                Notify("Download will resume after the other units have finished." & vbLf)
                deleg2.Invoke(unit)
                Exit Sub
            End If
        End If


        'Attempt to append corrected time to the E and I records
        'Note: unit time is returned in the format "Wed 28Jun2006 13:39:15" (with ends trimmed)

        Dim RT22 As ReferenceTime = unit.GetReferenceTime() 'RT22 is the reference time stored on disk
        Dim HTcheck As New TimeSpan(hostDateAvg.Ticks - RT22.HostTimeZero.Ticks)

        'MessageBox.Show(RT.UnitTimeZero & vbLf & RT.HostTimeZero, "Recorded Unit, Host Time")

        If HTcheck.Days < 365 Then     'a previous useful reference time was found

            Dim tDriftNum As Long = ((hostDateAvg.Ticks - unitDateAvg.Ticks) - _
                                    (RT22.HostTimeZero.Ticks - RT22.UnitTimeZero.Ticks))
            Dim tDriftDenom As Long = (hostDateAvg.Ticks - RT22.HostTimeZero.Ticks)
            Dim timeDriftRatio As Double = (tDriftNum / tDriftDenom)

            'MessageBox.Show(tDriftNum.ToString & vbLf & tDriftDenom.ToString & vbLf & timeDriftRatio.ToString, "Time Drift")

            For i As Integer = 0 To EList.Count - 1
                Dim subitem As String = EList.Item(i).Substring(7, 19)
                Dim itemTime As Date = Date.Parse(subitem)
                'MessageBox.Show(itemTime.ToString & vbLf & RT22.UnitTimeZero.ToString)

                If itemTime.Ticks >= RT22.UnitTimeZero.Ticks Then
                    'only perform time correction on data taken after the last 870 clock reset
                    Dim offsetZero As Long = RT22.HostTimeZero.Ticks - RT22.UnitTimeZero.Ticks
                    Dim Te_minus_To As Long = (itemTime.Ticks - RT22.UnitTimeZero.Ticks)

                    Dim drift As New TimeSpan(offsetZero + _
                    CLng(Math.Round(Te_minus_To * timeDriftRatio, 0, MidpointRounding.AwayFromZero)))

                    Dim correctedTime As Date = itemTime.Add(drift)
                    EList.Item(i) = EList.Item(i).TrimEnd(trimChars)
                    EList.Item(i) &= ", " & correctedTime.ToString("ddMMMyyyy HH:mm:ss.fff")
                End If
            Next

            For i As Integer = 0 To IList.Count - 1
                Dim subitem As String = IList.Item(i).Substring(7, 19)
                Dim itemTime As Date = Date.Parse(subitem)
                'MessageBox.Show(itemTime.ToString & vbLf & RT22.UnitTimeZero.ToString)

                If itemTime.Ticks >= RT22.UnitTimeZero.Ticks Then
                    'only perform time correction on data taken after the last 870 clock reset
                    Dim offsetZero As Long = RT22.UnitTimeZero.Ticks - RT22.HostTimeZero.Ticks
                    Dim Te_minus_To As Long = itemTime.Ticks - RT22.UnitTimeZero.Ticks

                    Dim drift As New TimeSpan(offsetZero + _
                    CLng(Math.Round(Te_minus_To * timeDriftRatio, 0, MidpointRounding.AwayFromZero)))

                    Dim correctedTime As Date = itemTime.Add(drift)
                    IList.Item(i) = IList.Item(i).TrimEnd(trimChars)
                    IList.Item(i) &= ", " & correctedTime.ToString("ddMMMyyyy HH:mm:ss.fff")
                    'MessageBox.Show(itemTime.ToString & vbLf & correctedTime.ToString, "Uncorrected, Corrected")
                End If
            Next
        End If



        'Store report data somewhere
        '(if the download routine made it this far, the data is good)

        unit.ErrorString = unitErrorString  'R98
        unit.EList = EList
        unit.IList = IList
        unit.NumExceedances = EList.Count
        unit.NumIntervals = IList.Count
        unit.BattVoltage = battVoltage
        unit.LastDL = hostDateAvg
        unit.NumStartStops = numStartStops
        unit.NumCalibrations = numCalibrations
        unit.CalLevel = calLevel
        If unit.IncludeR Then
            unit.RList = RList
        End If
        If unit.IncludeQ Then
            unit.QList = QList
        End If
        If unit.IncludeLC Then
            unit.LList = LList
            unit.CList = CList
        End If

        unit.SerializeReports()

        Dim sqlerror As String = unit.SaveToDB(RTcurrent)
        If Not sqlerror Like "" Then
            Notify(sqlerror & vbLf)
        End If



        'Reset unit time
        If unit.ResetTimeYN Then

            Write("M4" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If Not result Like "" Then
                Write("M4" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
            End If

            Dim newUnitTime As Date = Date.Now
            Write("S6," & newUnitTime.ToString("HH:mm:ss") & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If result Like "*Stop*" Then
                Write("M4" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M4" & vbCr)
                        result = ReadLine(CChar(vbLf))
                    End If
                End If
                newUnitTime = Date.Now
                Write("S6," & newUnitTime.ToString("HH:mm:ss") & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            If Not result Like "" Then
                newUnitTime = Date.Now
                Write("S6," & newUnitTime.ToString("HH:mm:ss") & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            Write("M3" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If Not result Like "" Then
                Write("M3" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M3" & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
            End If

            'take a new time difference, store it to disk to use to compute the time drift for future data
            'Get unit and host times just after the reset -
            'this will be the reference "Time Zero" for the time drift calculations

            Dim RTzero As ReferenceTime = GetUnitTime(worker, e)

            'Record the unit & host times
            unit.SerializeReferenceTime(RTzero)
            unit.ResetTimeYN = False

        End If


        'Reset unit data
        If unit.ResetDataYN AndAlso sqlerror = "" Then
            Write("M4" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If Not result Like "" Then
                Write("M4" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
            End If
            Write("S1,1" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If result Like "*Stop*" Then
                Write("M4" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M4" & vbCr)
                        result = ReadLine(CChar(vbLf))
                    End If
                End If
                Write("S1,1" & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            Write("S1,1" & vbCr)
            result = ReadLine(CChar(vbLf))


            'change exceedance threshold if necessary
            'Dim desiredExcdThresh As Integer = 0
            'Dim now As Date = Date.Now
            'Dim nowDay As DayOfWeek = now.DayOfWeek
            'Dim nowTime As TimeSpan = now.TimeOfDay
            'Dim nowtimesec As Long = CLng(nowTime.TotalSeconds)  'truncate the fractional seconds
            'nowTime = New TimeSpan(nowtimesec * 10000000)

            'Select Case nowDay
            '    Case DayOfWeek.Sunday
            '        desiredExcdThresh = unit.ExcdNight
            '    Case DayOfWeek.Saturday
            '        If Form1.appset.IsSatAWorkday Then
            '            If Form1.appset.WorkdayStart <= Form1.appset.WorkdayEnd Then
            '                If nowTime > Form1.appset.WorkdayStart AndAlso nowTime < Form1.appset.WorkdayEnd Then
            '                    desiredExcdThresh = unit.ExcdDay
            '                Else
            '                    desiredExcdThresh = unit.ExcdNight
            '                End If
            '            Else
            '                If nowTime > Form1.appset.WorkdayEnd AndAlso nowTime < Form1.appset.WorkdayStart Then
            '                    desiredExcdThresh = unit.ExcdNight
            '                Else
            '                    desiredExcdThresh = unit.ExcdDay
            '                End If
            '            End If
            '        Else
            '            desiredExcdThresh = unit.ExcdNight
            '        End If

            '        'otherwise
            '    Case Else
            '        If Form1.appset.WorkdayStart <= Form1.appset.WorkdayEnd Then
            '            If nowTime > Form1.appset.WorkdayStart AndAlso nowTime < Form1.appset.WorkdayEnd Then
            '                desiredExcdThresh = unit.ExcdDay
            '            Else
            '                desiredExcdThresh = unit.ExcdNight
            '            End If
            '        Else 'ex. day starts at 6am and ends at 1am
            '            If nowTime > Form1.appset.WorkdayEnd AndAlso nowTime < Form1.appset.WorkdayStart Then
            '                desiredExcdThresh = unit.ExcdNight
            '            Else
            '                desiredExcdThresh = unit.ExcdDay
            '            End If
            '        End If
            'End Select

            Dim desiredExcdThresh As Integer = 0
            If Not Form1.appset.StopAtNight Then
                'replace above with  Not Form1.appset.StopAtNight  later
                'Aberdeen only is test - soon all units won't run at night
                If Form1.IsDaytime Then
                    desiredExcdThresh = unit.ExcdDay
                Else
                    desiredExcdThresh = unit.ExcdNight
                End If
            Else
                desiredExcdThresh = unit.ExcdDay 'since unit won't run at night
            End If

            If desiredExcdThresh <> CInt(excdThreshTemp) AndAlso desiredExcdThresh > 29 Then
                If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                    Write("S64," & desiredExcdThresh.ToString & vbCr)
                    result = ReadLine(CChar(vbLf))
                    Write("S64," & desiredExcdThresh.ToString & vbCr)
                    result = ReadLine(CChar(vbLf))
                Else
                    Write("S64," & desiredExcdThresh.ToString & vbCr)
                    result = ReadLine(CChar(vbLf))
                    Write("S64," & desiredExcdThresh.ToString & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
                Write("Q64" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
                If result Like "*ARNING*" Then
                    Write("Q64" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim(trimChars)
                End If
                If result Like "*No response.*" Then  'ReadLine timed out
                    Dim tempd, tempd2 As Double

                    comx.DiscardInBuffer()
                    Write("Q64" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim(trimChars)
                    result2 = ReadLine(CChar(vbLf), 1)
                    result2 = result2.Trim(trimChars)
                    parseRes = Double.TryParse(result, tempd)
                    parseRes2 = Double.TryParse(result2, tempd2)
                    If parseRes2 Then       'use second read entry
                        excdThreshTemp = tempd2
                    ElseIf parseRes Then    'use first read entry
                        excdThreshTemp = tempd
                    End If
                Else
                    excdThreshTemp = Double.Parse(result)
                End If
                If desiredExcdThresh <> CInt(excdThreshTemp) AndAlso desiredExcdThresh > 29 Then
                    If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                        Write("S64," & desiredExcdThresh.ToString & vbCr)
                        result = ReadLine(CChar(vbLf))
                        Write("S64," & desiredExcdThresh.ToString & vbCr)
                        result = ReadLine(CChar(vbLf))
                    Else
                        Write("S64," & desiredExcdThresh.ToString & vbCr)
                        result = ReadLine(CChar(vbLf))
                        Write("S64," & desiredExcdThresh.ToString & vbCr)
                        result = ReadLine(CChar(vbLf))
                    End If
                End If
            End If

            Write("M3" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim()
            If Not result Like "" Then
                Write("M3" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M3" & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
            End If
        End If

        'store exceedance threshold
        unit.ExcdThreshold = CInt(excdThreshTemp)




        'Tell unit to dial out on both exceedance and alarm (such as low battery)
        If unit.AllowCallIns Then
            Write("Q156" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not String.Compare(result, "T" & modemPhoneNum) = 0 AndAlso Not unit.UnitNum = "L99" Then
                'test - don't reset phone # on unit 99
                Write("S156,T" & modemPhoneNum & vbCr)
                result = ReadLine(CChar(vbLf))
                Write("S156,T" & modemPhoneNum & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            Write("S155,3" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S155,3" & vbCr)
            result = ReadLine(CChar(vbLf))
        Else    'uncomment for in-house unit only
            Write("S155,0" & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S155,0" & vbCr)
            result = ReadLine(CChar(vbLf))
        End If



        'Set unit's run and stop timer (if necessary)
        If unit.TimerChanged Then
            Write("S24," & unit.TimerRun1 & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S24," & unit.TimerRun1 & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S25," & unit.TimerStop1 & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("S25," & unit.TimerStop1 & vbCr)
            result = ReadLine(CChar(vbLf))
            Write("Q24" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not result = unit.TimerRun1 Then
                Write("S24," & unit.TimerRun1 & vbCr)
                result = ReadLine(CChar(vbLf))
                Write("S24," & unit.TimerRun1 & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            Write("Q25" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            If Not result = unit.TimerStop1 Then
                Write("S25," & unit.TimerStop1 & vbCr)
                result = ReadLine(CChar(vbLf))
                Write("S25," & unit.TimerStop1 & vbCr)
                result = ReadLine(CChar(vbLf))
            End If
            unit.TimerChanged = False
        End If



        'Send custom parameters
        If unit.SendCustP Then
            Notify("*** Custom parameters ***" & vbLf)
            Dim seps() As Char = {CChar(vbLf), CChar(vbCr)}
            Dim cmds() As String = unit.CustParams.Split(seps, StringSplitOptions.RemoveEmptyEntries)
            For Each cmd As String In cmds
                cmd.Trim()
                Write(cmd & vbCr)
                result = ReadLine(CChar(vbLf))
                'result = result.Trim(trimChars)
                If Not (result Like "*" & vbLf & "*" OrElse result Like "*" & vbCr & "*") Then
                    Write(cmd & vbCr)
                    result = ReadLine(CChar(vbLf))
                End If
            Next
            Notify("*** End of custom parameters ***" & vbLf)
            unit.SendCustP = False
        End If



        'check to see if unit is running
        'if nighttime, stop unit (M4); otherwise, run unit (M3)

        Write("R3" & vbCr)
        result = ReadLine(CChar(vbLf))
        If Form1.appset.StopAtNight AndAlso Not Form1.IsDaytime Then   'stop unit
            If Not result Like "*Stop*" Then
                Write("M4" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M4" & vbCr)
                        result = ReadLine(CChar(vbLf))
                    End If
                End If
                Write("R3" & vbCr)
                result = ReadLine(CChar(vbLf))
                If Not result Like "*Stop*" Then
                    Write("M4" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M4" & vbCr)
                        result = ReadLine(CChar(vbLf))
                        result = result.Trim()
                        If Not result Like "" Then
                            Write("M4" & vbCr)
                            result = ReadLine(CChar(vbLf))
                        End If
                    End If
                End If
            End If
        Else  'run unit
            If Not result Like "*Run*" Then
                Write("M3" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim()
                If Not result Like "" Then
                    Write("M3" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M3" & vbCr)
                        result = ReadLine(CChar(vbLf))
                    End If
                End If
                Write("R3" & vbCr)
                result = ReadLine(CChar(vbLf))
                If Not result Like "*Run*" Then
                    Write("M3" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim()
                    If Not result Like "" Then
                        Write("M3" & vbCr)
                        result = ReadLine(CChar(vbLf))
                        result = result.Trim()
                        If Not result Like "" Then
                            Write("M3" & vbCr)
                            result = ReadLine(CChar(vbLf))
                        End If
                    End If
                End If
            End If
        End If



        'Store report data somewhere
        '(if the download routine made it this far, the data is good)

        unit.ErrorString = unitErrorString  'R98
        unit.EList = EList
        unit.IList = IList
        unit.NumExceedances = EList.Count
        unit.NumIntervals = IList.Count
        unit.BattVoltage = battVoltage
        unit.LastDL = hostDateAvg
        unit.NumStartStops = numStartStops
        unit.NumCalibrations = numCalibrations
        'unit.ExcdThreshold = CInt(excdThreshTemp)
        unit.CalLevel = calLevel



        'Download complete
        Write("M12" & vbCr)
        Notify("*** Download of unit " & unit.UnitNum & " is complete ***" & vbCr)
        unit.IsDownloadInProgress = False
        unit.IsDownloadDone = True
        unit.DLTries = 0
        unit.DialTries = 0
        If Not sqlerror Like "" Then
            unit.IsDownloadDone = False
        End If
        If Not (unit.UnitNum = "null" OrElse unit.UnitNum = "temp") Then
            'If unit.IsDownloadDone = True AndAlso Me.modemMode = Mode.Dial Then
            If Form1.appset.DialInProg Then 'remove unit from the status bar lists - even if it dialed in
                Dim SBarEA As New StatusBarEventArgs(unit.UnitNum, StatusBarEventArgs.SBAction.Remove)
                'RaiseEvent UpdateStatusBar(Me, SBarEA)
                RaiseEvent SBUnitCounter(Me, SBarEA)
                RaiseEvent UpdateMessages(Me, SBarEA)
            End If
            'If Me.modemMode = Mode.Listen Then  'make the unit available to the list
            '    unit.IsDownloadDone = False
            '    unit.IsDownloadInProgress = False
            'End If
            deleg2.Invoke(unit)
        End If
    End Sub

    Public Sub ReconfigureUnit(ByVal unit As LDUnit, ByVal askPass As Boolean, _
    ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs)

        'Reconfigures the report settings on the unit

        Dim result As New List(Of String)
        Dim unitnumstr As String = unit.UnitNum.Substring(1)    'remove the leading "L"
        Dim unitNum As Integer = Integer.Parse(unitnumstr)
        Dim isConfigGood As Boolean = True

        If askPass Then
            unit = getPass(unit, worker, e)
        End If

        Notify("Reconfiguring report settings..." & vbLf)
        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
            'set up Aberdeen units
            Write("M4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S1,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S37,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S40,2" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S44,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S61,200" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S62,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S69,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S71,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S78,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S80,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S88,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S89,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S90,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S91,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S92,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S93,40" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S94,90" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S95,4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S96,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S97,70" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S98,120" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S99,4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S100,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S101,110" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S102,160" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S103,4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S104,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S105,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S106,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S107,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S108,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S109,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S110,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S111,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S113,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S118,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S119,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S120,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S121,Wind" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S122,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S123,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S124,Knot" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S125,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S140,RH" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S149,008.1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S150,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S151,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S152,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S153,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S155,3" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S156,14102721390" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S157," & unitNum & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            'Write("s158,X4 E0 Q0 V0 T M1 S0=5 S10=100 &D3" & vbCr)
            'result.add(readline(cchar(vblf)))
            Write("S159,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S160,10.8" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S161,00:03" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S167,0406002701890A3D0666" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S168,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S169,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S170,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S171,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S172,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S173,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S174,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S192,2" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S193,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S194,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S195,2" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S196,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S197,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            'Write("S201,2" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S202,0" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S207,8" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S210,49" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            Write("M3" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
        Else
            'set up CERL units
            Write("M4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S1,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S12,5" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S16,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S38,06:00" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))

            Write("S61,200" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S62,200" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S63,100" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S64,200" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S69,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S71,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S78,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S80,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S88,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S89,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S90,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S91,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S92,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S96,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S99,4" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S100,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S104,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S105,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S106,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S107,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S108,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S109,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S110,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S111,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S113,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S118,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S119,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S120,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S150,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S151,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S152,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S153,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S155,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S160,09.5" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S168,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S169,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S170,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S171,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S172,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S173,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S174,1" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S192,2" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S193,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S194,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S195,2" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S196,0" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Write("S197,65535" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            'Write("S201,2" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S202,0" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S207,008.00" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            'Write("S210,49" & vbCr)
            'result.Add(ReadLine(CChar(vbLf)))
            Write("M3" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
        End If

        If worker.CancellationPending Then
            Write("M12" & vbCr)
            result.Add(ReadLine(CChar(vbLf)))
            Throw New CancelException
        End If

        'Write("" & vbCr)
        'result.Add(ReadLine(CChar(vbLf)))

        For Each item As String In result
            item.Trim()
            If item.Length > 4 Then     'an error occurred
                isConfigGood = False
            End If
        Next

        If isConfigGood = False Then
            ReconfigureUnit(unit, False, worker, e)   'repeat the configuration
        End If

        unit.Initialize = False
    End Sub

#Region "Helper Methods"

    Public Function getPass(ByVal unit As LDUnit, _
    ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs) As LDUnit

        'unified way to get unit password
        Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)

        Dim result As String = ""
        Dim result2 As String = ""
        Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab)}
        Dim checkResponse As Boolean = False


        For i As Integer = 1 To 10
            result = ReadLine(":"c, 2)
            result = result.Trim(trimChars)
            'MessageBox.Show(result, "Read result")
            If result Like "*870?????*" Then
                Dim pos As Integer = result.IndexOf("870")
                Dim pass As String = result.Substring(pos, 8)
                unit.UnitSerial = pass
                Write(pass & ":" & unit.LockCode & vbCr)
                result = ReadLine(CChar(vbLf), 2)
                If result Like "*Ready*" Then
                    Exit For
                End If
            ElseIf result Like "*ARNING*" Then      'a CERL unit
                checkResponse = True
                Exit For
            ElseIf result Like "*E*" OrElse result Like "*I*" Then
                Write("P999" & vbCr)
                'checkResponse = True
                Exit For
            ElseIf i >= 10 Then  'see if unit is responsive
                checkResponse = True
            End If
        Next

        If checkResponse Then
            comx.DiscardInBuffer()
            Write("R1" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
            Write("R1" & vbCr)
            result = ReadLine(CChar(vbLf))
            result = result.Trim(trimChars)
```

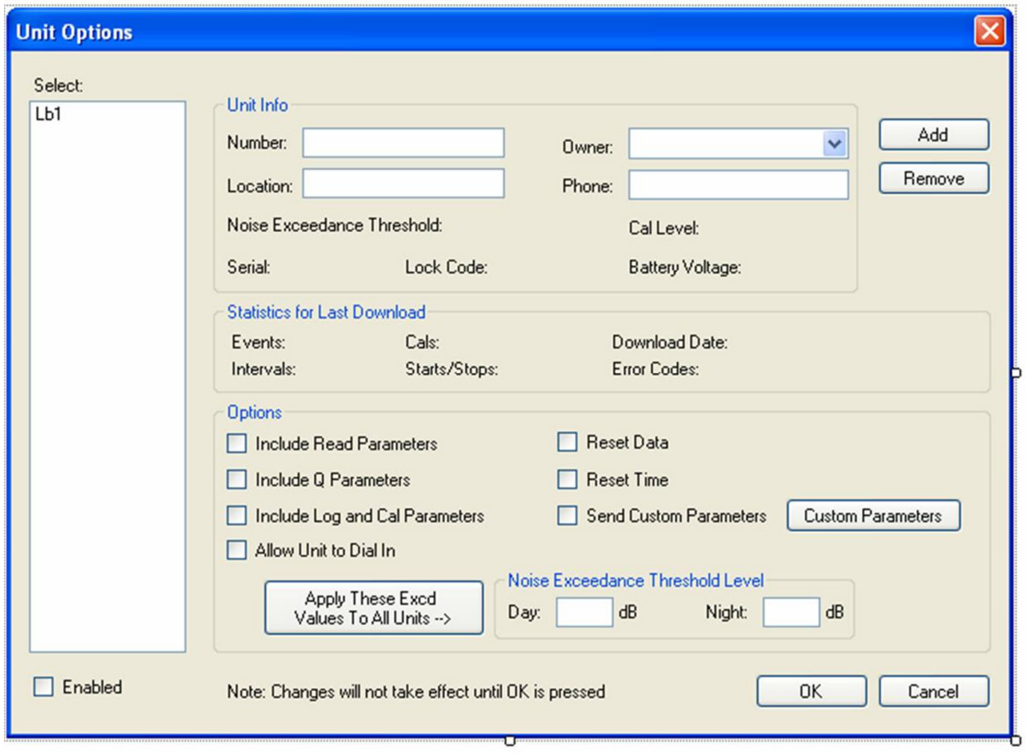
```vb
            If Not result Like "*No response.*" Then  'unit is responsive - get unit serial
                'result Like "*Larson*" OrElse result Like "*ARNING*" Then  'get unit serial #
                comx.DiscardInBuffer()
                Write("R89" & vbCr)
                result = ReadLine(CChar(vbLf))
                result = result.Trim(trimChars)
                If result Like "*No response.*" Then  'ReadLine timed out
                    comx.DiscardInBuffer()
                    Write("R89" & vbCr)
                    result = ReadLine(CChar(vbLf))
                    result = result.Trim(trimChars)
                    result2 = ReadLine(CChar(vbLf), 1)
                    result2 = result2.Trim(trimChars)
                    If Not result Like "" OrElse Not result Like "*ARNING*" Then       'use first read entry
                        unit.UnitSerial = "870" & result.Trim(trimChars)
                    ElseIf Not result2 Like "" Then    'use second read entry
                        unit.UnitSerial = "870" & result2.Trim(trimChars)
                    Else
                        Throw New FormatException("Can't read from unit - will try again later")
                    End If
                Else
                    unit.UnitSerial = "870" & result.Trim(trimChars)
                End If
            Else    'unit is not responsive - disconnect
                unit.IsDownloadInProgress = False
                Throw New Exception("Could not log onto unit")
            End If
        End If

        Return unit

    End Function

    Public Function GetUnitTime(ByVal worker As BackgroundWorker, ByVal e As DoWorkEventArgs) As ReferenceTime
        'Get the unit time and the PC time to check the time difference
        'take 8 samples, then average
        'The average represents a snapshot of the measured unit and computer time, taken
        '   at a single point in time
        'The averaging is done solely to reduce the influence of communications latency on the difference
        '   between the recorded unit time and the recorded computer time

        'Note: unit time is returned in the format "Wed 28Jun2006 13:39:15" (with ends trimmed)

        Dim result As String = ""
        Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab)}
        Dim tDifferenceTemp(7) As Double
        Dim unitDateTicks As Long = 0
        Dim hostDateTicks As Long = 0
        Dim unitDate(7) As Date
        Dim hostDate(7) As Date
        Dim tOffsetAvg As Double = 0
        Dim tryp() As Boolean = {False, False, False, False, False, False, False, False}
        Dim proceed As Boolean = True

        'take 4 samples
        For i As Integer = 0 To 3
            Dim tDifferenceTemp2 As TimeSpan = New TimeSpan(0)

            'unitdate(i) = New Date(1)

            'The host time is captured before the unit time is received, because Ed has observed that the
            'sending latency of the command sent to the 870 is less than the receiving latency of the response
            'sent back to the computer


            Write("R2" & vbCr)
            hostDate(i) = Date.Now      'immediately get computer time
            result = ReadLine(CChar(vbLf), 5) 'read host time



            result = result.Trim(trimChars)

            If result Like "*No response.*" OrElse result.Length > 30 Then   'readline read an invalid date string
                tryp(i) = False
                Exit For 'and checker will get the time again
            ElseIf result Like "*\?\?\?*" Then
                'no date is entered - enter today's date and day of week
                Write("M4" & vbCr)
                ReadLine(CChar(vbLf))
                Write("S7," & Date.Now.ToString("MM/dd/yy") & vbCr)
                ReadLine(CChar(vbLf))
                Write("S8," & Date.Now.DayOfWeek + 1 & vbCr)
                ReadLine(CChar(vbLf))
                Write("M3" & vbCr)
                ReadLine(CChar(vbLf))
            ElseIf Not result Like "#*" Then
                result = result.Remove(0, 4)    'trim off the day of week from the date - it is redundant
                '                                   and will introduce errors if it was set wrong on the 870
            End If

            tryp(i) = Date.TryParse(result, unitDate(i))
            hostDate(i) = hostDate(i).AddMilliseconds(45)   'assume 45 msec transmission delay -
            'this is based on the delay for the NIST ACTS timekeeping service

            tDifferenceTemp2 = unitDate(i).Subtract(hostDate(i))
            tDifferenceTemp(i) = tDifferenceTemp2.TotalSeconds 'Mod 3600    'ignore the difference in hours ↙
    -
            '                                                           prevents time zone differences from
            '                                                           screwing up the recorded time

        Next

        'wait .5 secs
        Thread.Sleep(500)

        'take 4 more samples
        For i As Integer = 4 To 7
            Dim tDifferenceTemp2 As TimeSpan = New TimeSpan(0)

            'unitDate(i) = New Date(1)

            'The host time is captured before the unit time is received, because Ed has observed that the
            'sending latency of the command sent to the 870 is less than the receiving latency of the      ↙
    response
            'sent back to the computer


            Write("R2" & vbCr)
            hostDate(i) = Date.Now      'immediately get computer time
            result = ReadLine(CChar(vbLf), 5) 'read host time



            result = result.Trim(trimChars)
            If result Like "*No response.*" OrElse result.Length > 30 Then
                tryp(i) = False
                Exit For 'and checker will get the time again
            ElseIf Not result Like "#*" Then
                result = result.Remove(0, 4)    'trim off the day of week from the date - it is redundant
                '                                    and will introduce errors if it was set wrong on the 870
            End If

            'UnitDate(i) is passed to tryparse by reference, and thus is edited directly by this method
            tryp(i) = Date.TryParse(result, unitDate(i))
            hostDate(i) = hostDate(i).AddMilliseconds(45)   'assume 45 msec transmission delay -
            'this is based on the delay for the NIST ACTS timekeeping service

            tDifferenceTemp2 = unitDate(i).Subtract(hostDate(i))
            tDifferenceTemp(i) = tDifferenceTemp2.TotalSeconds 'Mod 3600    'ignore the difference in hours ↙
    -
            '                                                           prevents time zone differences from
            '                                                           screwing up the recorded time
        Next

        'find the average time and offset for the first three trials
        'shifting bits to the right by 3 is the same as dividing by 8

        For i As Integer = 0 To 7
            unitDateTicks += unitDate(i).Ticks >> 3
            hostDateTicks += hostDate(i).Ticks >> 3
            tOffsetAvg += tDifferenceTemp(i) * 0.125
        Next

        Dim RT As New ReferenceTime(New Date(unitDateTicks), New Date(hostDateTicks), tOffsetAvg)

        'check values
        'MessageBox.Show(RT.UnitTimeZero & vbLf & RT.HostTimeZero, "Time Check")

        'Sometimes the 870 will not receive a sent command, or the program will time out before
        'the 870's response is received. This condition will result in only 7 responses received
        'for the 8 commands sent. This routine needs to verify that 8 responses were received
        'so that it doesn't return an errorneous unit time value.

        'if less than 8 responses were received, the tOffsetAvg will be very large, on the order of years
        '(1 year = 31557600 seconds)

        For i As Integer = 0 To 7
            'if any tryp(i) is false, get time again
            If tryp(i) = False Then
                proceed = False
            End If
        Next

        'let user break this loop, in case the program gets stuck (the corrupted RT won't be saved)
        If worker.CancellationPending Then
            e.Cancel = True
            Write("M12" & vbCr)
            result = ReadLine(CChar(vbLf))
            Throw New CancelException()
        End If

        If tOffsetAvg > 31557600 OrElse proceed = False Then
            Notify("." & vbLf)
            Me.getUnitTimeCounter += 1
            If Me.getUnitTimeCounter > 10 Then
                Me.getUnitTimeCounter = 0
                Throw New FormatException("Can't read from unit - will try again later")
            End If
            RT = GetUnitTime(worker, e)  'run until RT is correct - last RT will be returned to the calling ↙
    program
        End If

        Return RT

    End Function

    Private Shared Function IsE(ByVal line As String) As Boolean
        If line.StartsWith("E") AndAlso line.Length >= 153 AndAlso NumCommas(line) = 18 Then
            'try to parse each cell - takes lots of CPU time,
            'but is needed to ensure the data will go into the database smoothly
            'note that the corrected time has not been appended to these records yet - they will be 19 cols ↙
    wide
            Dim sep() As Char = {","c}
            Dim cells() As String = line.Split(sep, StringSplitOptions.None)
            Dim tryp(cells.Length - 1) As Boolean
            Dim trypFinal As Boolean = True
            Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab), "-"c}

            For i As Integer = 0 To cells.Length - 1
                cells(i) = cells(i).Trim(trimChars)
            Next
            tryp(0) = True
            For i As Integer = 1 To tryp.Length - 1
                tryp(i) = False
            Next
            tryp(1) = Date.TryParse(cells(1), New Date)
            If cells(2).Length < 11 Then
                tryp(2) = True
            End If
            For i As Integer = 3 To 7
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next
            For i As Integer = 8 To 9
                tryp(i) = Integer.TryParse(cells(i), New Integer)
            Next
            If cells(10).Length < 16 Then
                tryp(10) = True
            End If
            For i As Integer = 11 To 13
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next
            If cells(14).Length < 4 Then
                tryp(14) = True
            End If
            For i As Integer = 15 To 18
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next

            For i As Integer = 0 To tryp.Length - 1
                If tryp(i) = False Then
                    trypFinal = False
                End If
            Next

            If trypFinal = True Then
                Return True
            Else
                Return False
            End If
        Else
            Return False
        End If
    End Function
    Private Shared Function IsI(ByVal line As String) As Boolean
        If line.StartsWith("I") AndAlso line.Length >= 237 AndAlso NumCommas(line) = 33 Then
            'try to parse each cell - takes lots of CPU time,
            'but is needed to ensure the data will go into the database smoothly
            'note that the corrected time has not been appended to these records yet - they will be 34 cols ↙
    wide
            Dim sep() As Char = {","c}
            Dim cells() As String = line.Split(sep, StringSplitOptions.None)
            Dim tryp(cells.Length - 1) As Boolean
            Dim trypFinal As Boolean = True
            Dim trimChars As Char() = {CChar(vbCr), CChar(vbLf), "%"c, " "c, CChar(vbTab), "-"c}

            For i As Integer = 0 To cells.Length - 1
                cells(i) = cells(i).Trim(trimChars)
            Next
            tryp(0) = True
            For i As Integer = 1 To tryp.Length - 1
                tryp(i) = False
            Next
            tryp(1) = Date.TryParse(cells(1), New Date)
            If cells(2).Length < 11 Then
                tryp(2) = True
            End If
            For i As Integer = 3 To 8
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next
            For i As Integer = 9 To 13
                tryp(i) = Integer.TryParse(cells(i), New Integer)
            Next
            tryp(14) = Double.TryParse(cells(14), New Double)
            tryp(15) = Integer.TryParse(cells(15), New Integer)
            tryp(16) = Double.TryParse(cells(16), New Double)
            tryp(17) = Integer.TryParse(cells(17), New Integer)
            tryp(18) = Double.TryParse(cells(18), New Double)
            tryp(19) = Integer.TryParse(cells(19), New Integer)
            tryp(20) = Double.TryParse(cells(20), New Double)
            tryp(21) = Integer.TryParse(cells(21), New Integer)
            tryp(22) = Double.TryParse(cells(22), New Double)
            tryp(23) = Integer.TryParse(cells(23), New Integer)
            For i As Integer = 24 To 26
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next
            If cells(27).Length < 4 Then
                tryp(27) = True
            End If
            For i As Integer = 28 To 33
                tryp(i) = Double.TryParse(cells(i), New Double)
            Next

            For i As Integer = 0 To tryp.Length - 1
                If tryp(i) = False Then
                    trypFinal = False
                End If
            Next
            If trypFinal = True Then
                Return True
            Else
                Return False
            End If
        Else
            Return False
        End If
    End Function
    Private Shared Function IsR(ByVal line As String) As Boolean
        If line.StartsWith("R") AndAlso NumCommas(line) > 0 Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Shared Function IsQ(ByVal line As String) As Boolean
        If line.StartsWith("Q") AndAlso NumCommas(line) > 0 Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Shared Function IsL(ByVal line As String) As Boolean
        If line.StartsWith("L") AndAlso NumCommas(line) = 5 Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Shared Function IsC(ByVal line As String) As Boolean
        If line.StartsWith("C") AndAlso NumCommas(line) = 4 Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Shared Function NumCommas(ByVal line As String) As Integer

        Dim re As New Regex(",")
        Dim mc As MatchCollection = re.Matches(line)

        Return mc.Count
    End Function
    Private Shared Function IsBadEntry(ByVal line As String) As Boolean
        'tests to see if line is a bad entry

        If line.Length < 2 Then     'blank line
            Return False
        Else
            If (line.StartsWith("E") AndAlso NumCommas(line) = 18) OrElse _
            (line.StartsWith("I") AndAlso NumCommas(line) = 33) OrElse _
            (line.StartsWith("R") AndAlso NumCommas(line) > 0) OrElse _
            (line.StartsWith("Q") AndAlso NumCommas(line) > 0) OrElse _
            (line.StartsWith("L") AndAlso NumCommas(line) = 5) OrElse _
            (line.StartsWith("C") AndAlso NumCommas(line) = 4) Then
                Return False
            Else        'line wasn't added to one of the lists
                Return True
            End If
        End If
    End Function

#End Region

#Region "Junk Methods and Vars"


    'Public Sub Reconfigure(ByVal lnum As String, ByVal worker As BackgroundWorker, ByVal e As 
    DoWorkEventArgs)
    '    'Dials a unit to reconfigure it only

    '    If modemMode = Mode.Listen Then
    '        Exit Sub
    '    End If

    '    Dim result As String = ""
    '    Dim deleg As New GetUnitDelegate(AddressOf Form1.GetUnit)
    '    Dim deleg2 As New ReturnUnitDelegate(AddressOf Form1.ReturnUnit)
    '    Dim unit As LDUnit = deleg.Invoke(lnum)


    '    If String.Compare(unit.UnitNum, "null") = 0 Then
    '        Exit Sub
    '    End If

    '    Notify("*** Now dialing unit " & unit.UnitNum & " ***" & vbCr)

    '    Try
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        comx.Open()
    '        comx.DiscardInBuffer()  'flush receive buffer

    '        Write("AT" & vbCr)
    '        result = ReadLine(CChar(vbLf))
    '        result &= ReadLine(CChar(vbLf))
    '        Write("ATZ" & vbCr)
    '        result = ReadLine(CChar(vbLf))
    '        result &= ReadLine(CChar(vbLf))
    '        'MessageBox.Show(result, "result after first write")
    '        If Not result Like "*OK*" Then
    '            comx.DiscardInBuffer()
    '            Write("AT" & vbCr)
    '            result = ReadLine(CChar(vbLf))
    '            result &= ReadLine(CChar(vbLf))
    '            'MessageBox.Show(result, "result after second AT command")
    '            If Not result Like "*OK*" Then
    '                Throw New NoModemException("Modem on " & comx.PortName & " not present or responsive")
    '            End If
    '        End If
    '        Write("ATDT" & unit.UnitPhoneNum & vbCr)
    '        result = ReadLine(CChar(vbLf), 90)
    '        result &= ReadLine(CChar(vbLf), 90)
    '        If result Like "*OK*" Then
    '            result &= ReadLine(CChar(vbLf), 90)
    '        End If
    '        If Not result Like "*CONNECT*" Then
    '            Write("atz" & vbCr)
    '            result = ReadLine(CChar(vbLf))
    '            result &= ReadLine(CChar(vbLf))
    '            Write("ATDT" & unit.UnitPhoneNum & vbCr)
    '            result = ReadLine(CChar(vbLf), 90)
    '            result &= ReadLine(CChar(vbLf), 90)
    '            If Not result Like "*CONNECT*" Then
    '                Throw New NoConnectException("Unable to connect to remote modem")
    '            End If
    '        End If

    '        If unit.UnitOwner = LDUnit.Owner.CERL Then  'limit modem speed to 14400 bps
    '            If comx.PortName = "COM1" Then
    '                Write("AT$MB14400 \N3" & vbCr)
    '                result = ReadLine(CChar(vbLf))
    '                result &= ReadLine(CChar(vbLf))
    '            Else
    '                Write("AT+MS=V32B,1,300,14400,300,14400" & vbCr)
    '                result = ReadLine(CChar(vbLf))
    '                result &= ReadLine(CChar(vbLf))
    '                Write("AT\N3" & vbCr)
    '                result = ReadLine(CChar(vbLf))
    '                result &= ReadLine(CChar(vbLf))
    '            End If
    '        End If

    '        unit = getPass(unit, worker, e)
    '        If unit.UnitNum = "null" Then
    '            Exit Sub
    '        End If

    '        ReconfigureUnit(unit)  'reconfigures unit on same thread

    '        Write("M12" & vbCr)
    '        result = ReadLine(CChar(vbLf), 10)
    '        'result &= ReadLine(CChar(vbLf), 10)
    '        Write("+++ath" & vbCr)
    '        result = ReadLine(CChar(vbLf))
    '        result &= ReadLine(CChar(vbLf))

    '        comx.Close()

    '    Catch ex As NoModemException    'don't use this modem to download the remaining units
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        Notify("Error: " & ex.msg & vbLf)
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        deleg2.Invoke(unit)
    '        Exit Sub
    '    Catch ex As NoConnectException      'what happens when the line is busy or unavailable
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        Notify("Error: " & ex.msg & vbLf)
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        unit.DialTries += 1
    '        If unit.DialTries > 6 Then
    '            'stop trying to download unit
    '            unit.IsDownloadDone = True
    '            Notify("Attempted to dial unit " & unit.UnitNum & " seven times." & vbLf)
    '            Notify("Will not try again until the next Dial command is issued." & vbLf)
    '        End If
    '        deleg2.Invoke(unit)
    '        Dial(worker, e)
    '    Catch ex As CancelException     'user cancels the operation - close the port and exit
    '        Notify(ex.msg & vbLf)
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        deleg2.Invoke(unit)
    '        e.Cancel = True
    '        Exit Sub
    '    Catch ex As InvalidOperationException      'port is closed when attempting to write data
    '        Notify("InvalidOperationException" & vbLf)
    '        Notify("Error: " & ex.Message & vbLf)
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        deleg2.Invoke(unit)
    '        Dial(worker, e)
    '    Catch ex As IO.IOException      'port is closed when attempting to write data
    '        Notify("IOException" & vbLf)
    '        Notify("Error: " & ex.Message & vbLf)
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        deleg2.Invoke(unit)
    '        Dial(worker, e)
    '    Catch ex As Exception
    '        'will usually be from the download routine, and result from a dropped
    '        'or error-prone connection

    '        Write("M12" & vbCr)     'tell 870 to shut down
    '        result = ReadLine(CChar(vbLf), 10)
    '        'result &= ReadLine(CChar(vbLf), 10)
    '        Write("+++ath" & vbCr)
    '        result = ReadLine(CChar(vbLf))
    '        result &= ReadLine(CChar(vbLf))
    '        If comx.IsOpen Then
    '            comx.Close()
    '        End If
    '        Notify("Error: " & ex.Message & vbLf)
    '        unit.IsDownloadInProgress = False
    '        unit.IsDownloadDone = False
    '        unit.DialTries += 1
    '        If unit.DialTries > 6 Then
    '            'stop trying to download unit
    '            unit.IsDownloadDone = True
    '            Notify("Attempted to dial unit " & unit.UnitNum & " seven times." & vbLf)
    '            Notify("Will not try again until the next Dial command is issued." & vbLf)
    '        End If
    '        deleg2.Invoke(unit)
    '    End Try

    'End Sub




    'Public Sub UpdateBuffer() 'ByVal value As String)
    '    rxBuffer.Append(comx.ReadExisting) 'value)

    'End Sub

    'Public Delegate Sub UpdateBufferDelegate() 'ByVal value As String)

    'Public Sub ReadPort(ByVal sender As Object, ByVal e As SerialDataReceivedEventArgs)
    '    'This handler will be added and removed at will by the program

    '    'The DataReceived event fires when the data in the serial port buffer exceeds the
    '    'ReceivedBytesThreshold value
    '    'The default ReceivedBytesThreshold value is 1 byte

    '    Dim deleg As UpdateBufferDelegate

    '    deleg = New UpdateBufferDelegate(AddressOf UpdateBuffer)
    '    deleg.Invoke() 'comx.ReadExisting)
    'End Sub

    'Public Function ReadUntilChar(ByVal searchChar As Char) As String
    '    'reads port until searchChar (usually vbLf) is found
    '    'returns the string of chars in the rxBuffer between the time this method was invoked
    '    'and searchChar was found
    '    'if timeout exceeded, just returns the characters received since this method was invoked,
    '    'and lets the calling method figure out what went wrong
    '    'Only use when ReadLine doesn't return a value

    '    Dim index As Integer = -1    'start index
    '    Dim index2 As Integer = -1   'end index
    '    Dim intermediateIndex As Integer = -1
    '    Dim countSinceLastCharReceived As Integer = 0  'timeout mechanism
    '    Dim lastChar As Char = "|"c
    '    Dim returnSB As New StringBuilder(100)
    '    Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
    '        New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)


    '    rxBuffer.Append("<br>")     'delineates a new command will be sent, prevents method from returning ↙
    a false positive
    '    index = rxBuffer.Length - 1 'because the last index is always length-1
    '    intermediateIndex = index

    '    'For debugging
    '    'MessageBox.Show(rxBuffer.ToString, "rxBuffer")

    '    'loop until the search char is found or until no new data has been received for the timeout count
    '    Do Until lastChar = searchChar OrElse countSinceLastCharReceived >= 300  '3 secs
    '        index2 = rxBuffer.Length - 1
    '        If intermediateIndex <> index2 Then
    '            intermediateIndex = index2
    '            countSinceLastCharReceived = 0  'reset counter if new data in rxbuffer
    '        Else
    '            countSinceLastCharReceived += 1 'increment counter if no new data
    '        End If
    '        lastChar = rxBuffer.Chars(index2)
    '        Threading.Thread.Sleep(10)  'pause for 10 ms - prevents this loop from hogging the computer
    '    Loop
    '    If index + 1 <= index2 Then     'data was received
    '        For i As Integer = index + 1 To index2
    '            returnSB.Append(rxBuffer.Chars(i))
    '        Next
    '    End If

    '    'update text box
    '    EA.AppendThisString = returnSB.ToString
    '    RaiseEvent UpdateOutBoxesEvent(Me, EA)

    '    Return returnSB.ToString

    'End Function

    'Public Function ReadUntilChar(ByVal searchChar As Char, ByVal timeout As Double) As String
    '    'Overloads ReadUntilChar, lets calling method set a timeout in seconds
    '    '(e.g. for connecting to a host modem)


    '    Dim index As Integer = -1    'start index
    '    Dim index2 As Integer = -1   'end index
    '    Dim intermediateIndex As Integer = -1
    '    Dim countSinceLastCharReceived As Integer = 0  'timeout mechanism
    '    Dim lastChar As Char = "|"c
    '    Dim returnSB As New StringBuilder(100)
    '    Dim time As Integer = CInt(timeout * 100)
    '    Dim EA As WindowsApplication1.UpdateOutBoxesEventArgs = _
    '        New UpdateOutBoxesEventArgs("", UpdateOutBoxesEventArgs.RW.Read)

    '    rxBuffer.Append("<br>")     'delineates a new command will be sent, prevents method from returning ↙
    a false positive
    '    index = rxBuffer.Length - 1 'because the last index is always length-1
    '    intermediateIndex = index

    '    'For debugging
    '    'MessageBox.Show(rxBuffer.ToString, "rxBuffer")

    '    'loop until the search char is found or until no new data has been received for the timeout count
    '    Do Until lastChar = searchChar OrElse countSinceLastCharReceived >= time
    '        index2 = rxBuffer.Length - 1
    '        If intermediateIndex <> index2 Then
    '            intermediateIndex = index2
    '            countSinceLastCharReceived = 0  'reset counter if new data in rxbuffer
    '        Else
    '            countSinceLastCharReceived += 1 'increment counter if no new data
    '        End If
    '        lastChar = rxBuffer.Chars(index2)
    '        Threading.Thread.Sleep(10)  'pause for 10 ms - prevents this loop from hogging the computer
    '    Loop

    '    For i As Integer = index + 1 To index2
    '        returnSB.Append(rxBuffer.Chars(i))
    '    Next

    '    'update text box
    '    EA.AppendThisString = returnSB.ToString
    '    RaiseEvent UpdateOutBoxesEvent(Me, EA)
    '    Return returnSB.ToString

    'End Function









    'Private timeOut As Integer = 1
    'Private timeOutCountUp As Boolean = True
    'Private newData As Integer = 1
    'Private newDataCountUp As Boolean = True
    'Public WithEvents timeoutTimer As Timers.Timer = New Timers.Timer(500) 'fire an event every 500 msec
    'Private txtOutBox As New Windows.Forms.TextBox

    'Public Sub TimeOutHandler(ByVal sender As Object, ByVal e As Timers.ElapsedEventArgs) _
    '    Handles timeoutTimer.Elapsed
    '    'this runs on a new thread - need to delegate the increment task back to the Modem thread
    '    Dim deleg As IncrementTimeOutDelegate = New IncrementTimeOutDelegate(AddressOf IncrementTimeOut)
    '    deleg.Invoke()
    'End Sub

    'Public Delegate Sub IncrementNewDataDelegate()
    'Public Delegate Sub IncrementTimeOutDelegate()

    'Public Sub IncrementTimeOut()
    '    If timeOut <= 0 Then
    '        timeOutCountUp = True
    '    ElseIf timeOut > 1500000000 Then
    '        timeOutCountUp = False      'count down
    '    End If

    '    If timeOutCountUp = True Then   'increment
    '        timeOut += 1
    '    Else                            'decrement
    '        timeOut = timeOut - 1
    '    End If
    'End Sub


    'Public Sub IncrementNewData()
    '    If newData <= 0 Then
    '        newDataCountUp = True
    '    ElseIf newData >= 1500000000 Then
    '        newDataCountUp = False      'count down
    '    End If

    '    If newDataCountUp Then   'increment
    '        newData += 1
    '    Else                            'decrement
    '        newData = newData - 1
    '    End If
    'End Sub

    'Public Function SendCommand(ByVal writeString As String, ByVal readTimeOut As Integer) As Integer
    '    'halts the calling method's execution until data is available at the serial port
    '    'if no data was found before specified timeout, returns false


    '    'Do Until localND <> newData OrElse localTO < timeOut - readTimeOut _
    '    '   OrElse localTO > timeOut + readTimeOut
    '    '    Threading.Thread.Sleep(25)  'pause 25 msec - keeps loop from eating all the processor time
    '    'Loop
    '    'Return False

    '    Dim index1 As Integer = rxBuffer.Length     'get current end position of receive buffer
    '    Dim index2 As Integer = 0
    '    'Dim localTO As Long = timeOut
    '    'Dim localND As Long = newData

    '    Write(writeString)

    '    'wait readTimeOut after the last data character is received

    '    Dim lastChar As Char = rxBuffer.Chars(rxBuffer.Length)

    'End Function

    'Public Function ReadUntil(ByVal searchPattern As String) As Boolean
    '    'reads input until the search string is found
    '    'use for procedural control when talking to the remote unit
    '    'possibly overload with a string array input parameter
    '    'return True if the string was found, false if string was not found
    '    'make sure to quit after a specified timeout

    '    Dim found As Boolean = False
    '    Dim isMoreText As Boolean = True

    '    rxBuffer.Remove(0, rxBuffer.Length)     'clear buffer

    '    Do While isMoreText
    '        '     isMoreText = Read(2)    '1 second timeout for more data to be received
    '        If Not isMoreText Then
    '            Exit Do
    '        End If
    '    Loop

    '    If rxBuffer.Length = 0 Then     'nothing read
    '        Return False
    '    End If

    '    found = rxBuffer.ToString Like searchPattern
    '    Return found

    'End Function





    'Dim msgBoxResult As Windows.Forms.DialogResult = _
    'MessageBox.Show("The following records from unit " & unit.UnitNum & " are invalid:" _
    '& vbCr & vbCr & badListStr & vbCr & vbCr & _
    '"These records have been written to the file " & unit.InstallPath & _
    '"Bad Records\error" & unit.UnitNum & " - " & _
    'hostDateAvg.ToLongDateString & ".txt" & vbCr & vbCr & _
    '"Abort: Reset the unit's memory and reconfigure report settings" & vbCr & _
    '"Retry: Try download again" & vbCr & _
    '"Ignore: Remove bad records from the report and continue", _
    '"Warning", _
    'MessageBoxButtons.AbortRetryIgnore, MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button2)

    'Select Case msgBoxResult
    '    Case DialogResult.Abort

    '        Write("R1" & vbCr)
    '        result = ReadLine(CChar(vbLf))
    '        If result Like "*Larson*" Then
    '            'Write("M4" & vbCr)
    '            'result = ReadLine(CChar(vbLf))
    '            'Write("S1,1" & vbCr)
    '            'result = ReadLine(CChar(vbLf))
    '            'Write("M3" & vbCr)
    '            'result = ReadLine(CChar(vbLf))
    '            ReconfigureUnit(unit)
    '        Else
    '            unit.IsDownloadInProgress = False
    '            unit.IsDownloadDone = False
    '            Notify("Warning! Unit " & unit.UnitNum & " has timed out." & vbCr)
    '            Notify("Download will resume after the other units have finished." & vbCr)

    '            'return unit to the LDUnitList,
    '            deleg2.Invoke(unit)
    '            Exit Sub
    '        End If
    '        downloadAgain = True

    '    Case DialogResult.Retry
    '        downloadAgain = True
    '    Case DialogResult.Ignore
    '        Exit Select
    'End Select



#End Region

End Class

Public Class NoModemException
    Inherits ApplicationException

    Protected m_msg As String

    Public Property msg() As String
        Get
            Return m_msg
        End Get
        Set(ByVal value As String)
            m_msg = value
        End Set
    End Property

    Public Sub New(ByVal message As String)
        msg = message
    End Sub

End Class

Public Class NoConnectException
    Inherits ApplicationException

    Protected m_msg As String

    Public Property msg() As String
        Get
            Return m_msg
        End Get
        Set(ByVal value As String)
            m_msg = value
        End Set
    End Property

    Public Sub New(ByVal message As String)
        msg = message
    End Sub

End Class

Public Class CancelException
    Inherits ApplicationException
    'thrown when user cancels an operation

    Protected m_msg As String

    Public Property msg() As String
        Get
            Return m_msg
        End Get
        Set(ByVal value As String)
            m_msg = value
        End Set
    End Property

    Public Sub New()
        msg = "Canceled by user."
    End Sub

    Public Sub New(ByVal message As String)
        msg = message
    End Sub

End Class

Public Class NotEnabledException
    Inherits ApplicationException
    'thrown when user cancels an operation

    Protected m_msg As String

    Public Property msg() As String
        Get
            Return m_msg
        End Get
        Set(ByVal value As String)
            m_msg = value
        End Set
    End Property

    Public Sub New()
        msg = "Unit is not enabled."
    End Sub

    Public Sub New(ByVal message As String)
        msg = message
    End Sub

End Class
```

Appendix D: Dial One Unit



```
Imports System.Windows.Forms

Public Class OneUnitDialog

    Private ulist As List(Of LDUnit)

    Private Sub OK_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles OK_Button
    .Click
        Me.DialogResult = System.Windows.Forms.DialogResult.OK

        'to only download one unit, mark all but the selected unit as having been downloaded
        If Cb1.SelectedIndex = 0 Then
            For Each unit As LDUnit In ulist
                unit.IsDownloadDone = False
                ci(unit)
            Next
        ElseIf Cb1.SelectedIndex = 1 Then
            For Each unit As LDUnit In ulist
                If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                    unit.IsDownloadDone = False
                    ci(unit)
                Else
                    unit.IsDownloadDone = True
                    ci(unit)
                End If
            Next
        ElseIf Cb1.SelectedIndex = 2 Then
            For Each unit As LDUnit In ulist
                If unit.UnitOwner = LDUnit.Owner.CERL Then
                    unit.IsDownloadDone = False
                    ci(unit)
                Else
                    unit.IsDownloadDone = True
                    ci(unit)
                End If
            Next
        Else
            Dim u As LDUnit = ulist.Item(Cb1.SelectedIndex - 3)
            ulist.RemoveAt(Cb1.SelectedIndex - 3)

            For Each unit As LDUnit In ulist
                unit.IsDownloadDone = True
            Next

            u.IsDownloadDone = False
            ci(u)
            ulist.Add(u)
        End If

        Me.Close()

    End Sub

    Private Sub ci(ByRef unit As LDUnit)
        If CheckBox1.Checked Then
            unit.Initialize = True
        Else
            unit.Initialize = False
        End If
    End Sub

    Private Sub Cancel_Button_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    Cancel_Button.Click
        Me.DialogResult = System.Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub

    Private Sub OneUnitDialog_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.
    FormClosingEventArgs) Handles Me.FormClosing
        Form1.ReturnAllUnits(ulist)

    End Sub

    Private Sub OneUnitDialog_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase
    .Load
        Dim cb1List As New ComboBox.ObjectCollection(Cb1)
        Dim ucomp As New uCompare()
        Dim comp As Collections.Generic.IComparer(Of LDUnit) = ucomp


        ulist = Form1.GetAllUnitsNoMark()

        Cb1.Items.Add("All Units")
        Cb1.Items.Add("Aberdeen")
        Cb1.Items.Add("CERL")

        ulist.Sort(comp)

        For Each unit As LDUnit In ulist
            Cb1.Items.Add(unit.UnitNum)
        Next


    End Sub
End Class

Public Class uCompare
    Implements Collections.Generic.IComparer(Of LDUnit)

    Public Function Compare(ByVal x As LDUnit, ByVal y As LDUnit) As Integer Implements System.Collections.
    Generic.IComparer(Of LDUnit).Compare
        Dim unit1 As LDUnit = DirectCast(x, LDUnit)
        Dim unit2 As LDUnit = DirectCast(y, LDUnit)

        Return String.Compare(unit1.UnitNum, unit2.UnitNum)
    End Function
End Class
```

Appendix E: Unit Options


Unit Options

Select:
Lb1

Unit Info
Number:
Owner:
Location:
Phone:
Noise Exceedance Threshold:
Cal Level:
Serial:
Lock Code:
Battery Voltage:

Add
Remove

Statistics for Last Download
Events:
Cals:
Download Date:
Intervals:
Starts/Stops:
Error Codes:

Options
☐ Include Read Parameters
☐ Reset Data
☐ Include Q Parameters
☐ Reset Time
☐ Include Log and Cal Parameters
☐ Send Custom Parameters
Custom Parameters
☐ Allow Unit to Dial In
Apply These Excd Values To All Units -->
Noise Exceedance Threshold Level
Day: dB
Night: dB

☐ Enabled

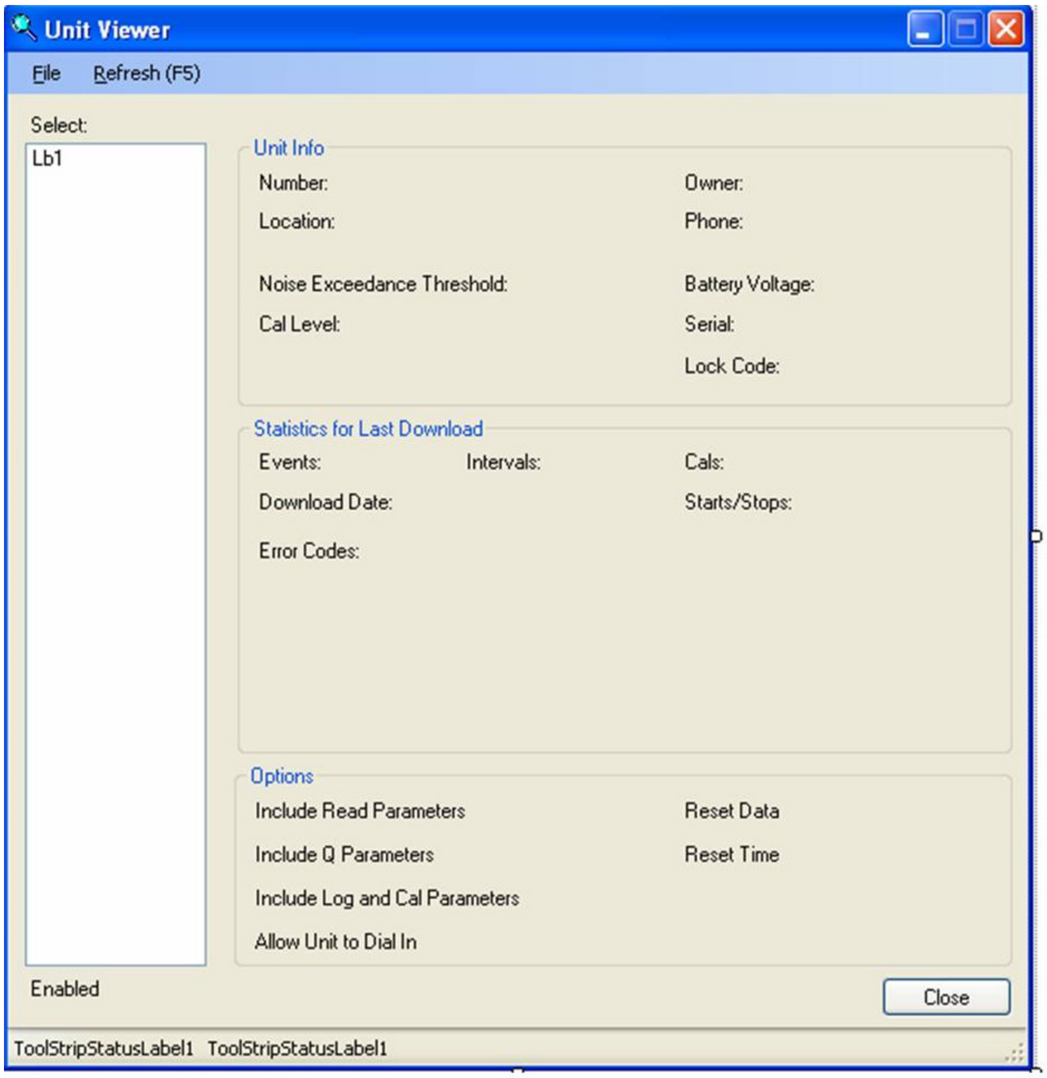
Note: Changes will not take effect until OK is pressed

OK
Cancel

```
Public Class UnitOptions

    Private workingUList As New List(Of LDUnit)
    Private uList As List(Of LDUnit)
    Private lb1OldSelect As Integer
    Private lb1NewSelect As Integer

    Protected t_unitSerial As String = ""
    Protected t_unitLNum As String = ""
    Protected t_unitLocation As String = ""
    Protected t_unitPhoneNum As String = ""
    Protected t_unitOwner As LDUnit.Owner
    Protected t_lockCode As String = ""
    Protected t_lastDL As DateTime

    Protected t_includeR As Boolean
    Protected t_includeLC As Boolean
    Protected t_includeQ As Boolean
    Protected t_allowCallIns As Boolean
    Protected t_resetDataYN As Boolean
    Protected t_resetTimeYN As Boolean
    Protected t_isEnabled As Boolean
    Protected t_sendCustP As Boolean

    Protected t_numExceedances As String = ""
    Protected t_numIntervals As String = ""
    Protected t_numStartStops As String = ""
    Protected t_numCalibrations As String = ""
    Protected t_battVoltage As String = ""
    Protected t_errorString As String = ""
    Protected t_excdThreshold As Integer = 0
    Public t_custParams As String = ""
    Protected t_calLevel As String = ""
    Protected t_excdDay As Integer = 0
    Protected t_excdNight As Integer = 0


    Private Sub UnitOptions_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase. ↙
    Load
        Dim ucomp As New uCompare()
        Dim comp As Collections.Generic.IComparer(Of LDUnit) = ucomp

        uList = Form1.GetAllUnits()

        'copy ulist to the working list
        workingUList.AddRange(uList)
        workingUList.Sort(comp)

        'populate the checked list box
        Lb1.BeginUpdate()
        Lb1.Items.Add("All Units")
        Lb1.Items.Add("Aberdeen")
        Lb1.Items.Add("CERL")

        For Each unit As LDUnit In workingUList
            Lb1.Items.Add(unit.UnitNum)
        Next
        Lb1.EndUpdate()

        lb1OldSelect = -1
        lb1NewSelect = -1

        Me.DialogResult = Windows.Forms.DialogResult.None
        Me.btnChangeExcd.Hide()

        tt.SetToolTip(Me.cbEn, "Indicates whether the program will dial out to the unit." _
        & vbLf & "Uncheck this box when you don't want the program to call the unit." & vbLf & vbLf & _
        "To stop a unit from calling in, uncheck ""Allow Unit to Dial In"".")
        tt.SetToolTip(Me.tbUN, "The ID (""L"") number of the remote unit.")
        tt.SetToolTip(Me.tbPhone, "The phone number of the remote unit.")
        tt.SetToolTip(Me.lblC, "The number of self-calibrations performed by the unit since the last data ↙
    reset.")
        tt.SetToolTip(Me.cbCallIn, "Check this box to make the unit call in." & vbLf & vbLf & _
        "Uncheck this box to make the unit stop calling in.")
        tt.SetToolTip(Me.tbExcdDay, "Sets the workday exceedance threshold level (in dB).")
        tt.SetToolTip(Me.tbExcdNight, "Sets the night and weekend exceedance" & vbLf & "threshold level (in ↙
    dB).")
        tt.SetToolTip(Me.Button1, "Creates a new unit, using the values entered in this window.")
        tt.SetToolTip(Me.Button3, "Deletes the currently selected unit.")

    End Sub


    Private Sub Lb1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ↙
    Lb1.SelectedIndexChanged
        lb1NewSelect = Lb1.SelectedIndex

        Select Case lb1OldSelect
            Case -1 'do nothing
            Case 0  'write option data to all units in working list
                For Each unit As LDUnit In workingUList
                    'unit.AllowCallIns = t_allowCallIns
                    unit.IncludeR = Me.t_includeR
                    unit.IncludeQ = Me.t_includeQ
                    unit.IncludeLC = Me.t_includeLC
                    unit.ResetDataYN = Me.t_resetDataYN
                    unit.ResetTimeYN = Me.t_resetTimeYN
                    'unit.IsEnabled = Me.t_isEnabled
                    unit.SendCustP = Me.t_sendCustP
                    If Me.t_sendCustP Then
                        unit.CustParams = Me.t_custParams
                        Form1.appset.CustParams = Me.t_custParams
                    End If
                Next
                Form1.appset.WorkdayExcdThresh = Me.t_excdDay
                Form1.appset.OtherExcdThresh = Me.t_excdNight
            Case 1  'write option data to Aberdeen units
                For Each unit As LDUnit In workingUList
                    If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                        unit.AllowCallIns = t_allowCallIns
                        unit.IncludeR = Me.t_includeR
                        unit.IncludeQ = Me.t_includeQ
                        unit.IncludeLC = Me.t_includeLC
                        unit.ResetDataYN = Me.t_resetDataYN
                        unit.ResetTimeYN = Me.t_resetTimeYN
                        'unit.IsEnabled = Me.t_isEnabled
                        unit.SendCustP = Me.t_sendCustP
                        If Me.t_sendCustP Then
                            unit.CustParams = Me.t_custParams
                            Form1.appset.CustParams = Me.t_custParams
                        End If
                    End If
                Next
            Case 2  'write option data to CERL units
                For Each unit As LDUnit In workingUList
                    If unit.UnitOwner = LDUnit.Owner.CERL Then
                        unit.AllowCallIns = t_allowCallIns
                        unit.IncludeR = Me.t_includeR
                        unit.IncludeQ = Me.t_includeQ
                        unit.IncludeLC = Me.t_includeLC
                        unit.ResetDataYN = Me.t_resetDataYN
                        unit.ResetTimeYN = Me.t_resetTimeYN
                        'unit.IsEnabled = Me.t_isEnabled
                        unit.SendCustP = Me.t_sendCustP
                        If Me.t_sendCustP Then
                            unit.CustParams = Me.t_custParams
                            Form1.appset.CustParams = Me.t_custParams
                        End If
                    End If
                Next
            Case Else   'write all form data to the appropriate unit
                With workingUList.Item(lb1OldSelect - 3)
                    .AllowCallIns = t_allowCallIns
                    .IncludeR = Me.t_includeR
                    .IncludeQ = Me.t_includeQ
                    .IncludeLC = Me.t_includeLC
                    .ResetDataYN = Me.t_resetDataYN
                    .ResetTimeYN = Me.t_resetTimeYN
                    .IsEnabled = Me.t_isEnabled
                    .UnitOwner = Me.t_unitOwner
                    .UnitPhoneNum = Me.t_unitPhoneNum
                    .UnitLocation = Me.t_unitLocation
                    .UnitNum = Me.t_unitLNum
                    .LockCode = Me.t_lockCode
                    .SendCustP = Me.t_sendCustP
                    If Me.t_sendCustP Then
                        .CustParams = Me.t_custParams
                    End If
                    .ExcdDay = Me.t_excdDay
                    .ExcdNight = Me.t_excdNight
                End With
        End Select

        Select Case lb1NewSelect
            Case -1 'do nothing
            Case 0  'gray out Unit Info, write data from the first unit to the form
                gbUnitInfo.Enabled = False
                gbStat.Enabled = False
                cbEn.Enabled = False
                gbExcd.Enabled = True
                Me.btnChangeExcd.Show()

                With Me
                    .t_includeR = workingUList.Item(0).IncludeR
                    .t_includeQ = workingUList.Item(0).IncludeQ
                    .t_includeLC = workingUList.Item(0).IncludeLC
                    .t_allowCallIns = workingUList.Item(0).AllowCallIns
                    .t_resetDataYN = workingUList.Item(0).ResetDataYN
                    .t_resetTimeYN = workingUList.Item(0).ResetTimeYN
                    .t_unitLNum = ""
                    .t_unitLocation = ""
                    .t_unitPhoneNum = ""
                    .t_unitOwner = LDUnit.Owner.Nobody
                    .t_unitSerial = ""
                    .t_isEnabled = workingUList.Item(0).IsEnabled
                    .t_sendCustP = workingUList.Item(0).SendCustP
                    .t_lockCode = ""
                    .t_battVoltage = ""
                    .t_numExceedances = ""
                    .t_numIntervals = ""
                    .t_numCalibrations = ""
                    .t_numStartStops = ""
                    .t_lastDL = New Date(1900, 1, 1, 0, 0, 0)
                    .t_errorString = ""
                    .t_custParams = Form1.appset.CustParams
                    .t_calLevel = ""
                    .t_excdDay = Form1.appset.WorkdayExcdThresh
                    .t_excdNight = Form1.appset.OtherExcdThresh
                End With
            Case 1  'gray out Unit Info, write data from the first APG unit to the form
                gbStat.Enabled = False
                gbUnitInfo.Enabled = False
                Dim idx As Integer = workingUList.FindIndex(AddressOf IsAberdeen)
                cbEn.Enabled = False
                gbExcd.Enabled = False
                Me.btnChangeExcd.Hide()

                With Me
                    .t_includeR = workingUList.Item(idx).IncludeR
                    .t_includeQ = workingUList.Item(idx).IncludeQ
                    .t_includeLC = workingUList.Item(idx).IncludeLC
                    .t_allowCallIns = workingUList.Item(idx).AllowCallIns
                    .t_resetDataYN = workingUList.Item(idx).ResetDataYN
                    .t_resetTimeYN = workingUList.Item(idx).ResetTimeYN
                    .t_unitLNum = ""
                    .t_unitLocation = ""
                    .t_unitPhoneNum = ""
                    .t_unitOwner = LDUnit.Owner.Aberdeen
                    .t_unitSerial = ""
                    .t_isEnabled = workingUList.Item(idx).IsEnabled
                    .t_sendCustP = workingUList.Item(idx).SendCustP
                    .t_lockCode = ""
                    .t_battVoltage = ""
                    .t_numExceedances = ""
                    .t_numIntervals = ""
                    .t_numCalibrations = ""
                    .t_numStartStops = ""
                    .t_lastDL = New Date(1900, 1, 1, 0, 0, 0)
                    .t_errorString = ""
                    .t_custParams = Form1.appset.CustParams
                    .t_calLevel = ""
                    .t_excdDay = -1
                    .t_excdNight = -1
                End With
            Case 2  'gray out Unit Info, write data from the first CERL unit to the form
                gbUnitInfo.Enabled = False
                gbStat.Enabled = False
                Dim idx As Integer = workingUList.FindIndex(AddressOf IsCerl)
                cbEn.Enabled = False
                gbExcd.Enabled = False
                Me.btnChangeExcd.Hide()

                With Me
                    .t_includeR = workingUList.Item(idx).IncludeR
                    .t_includeQ = workingUList.Item(idx).IncludeQ
                    .t_includeLC = workingUList.Item(idx).IncludeLC
                    .t_allowCallIns = workingUList.Item(idx).AllowCallIns
                    .t_resetDataYN = workingUList.Item(idx).ResetDataYN
                    .t_resetTimeYN = workingUList.Item(idx).ResetTimeYN
                    .t_unitLNum = ""
                    .t_unitLocation = ""
                    .t_unitPhoneNum = ""
                    .t_unitOwner = LDUnit.Owner.CERL
                    .t_unitSerial = ""
                    .t_isEnabled = workingUList.Item(idx).IsEnabled
                    .t_sendCustP = workingUList.Item(idx).SendCustP
                    .t_lockCode = ""
                    .t_battVoltage = ""
                    .t_numExceedances = ""
                    .t_numIntervals = ""
                    .t_numCalibrations = ""
                    .t_numStartStops = ""
                    .t_lastDL = New Date(1900, 1, 1, 0, 0, 0)
                    .t_errorString = ""
                    .t_custParams = Form1.appset.CustParams
                    .t_calLevel = ""
                    .t_excdDay = -1
                    .t_excdNight = -1
                End With
            Case Else   'load data from the selected unit to the class vars and the UI
                gbUnitInfo.Enabled = True
                gbStat.Enabled = True
                cbEn.Enabled = True
                gbExcd.Enabled = True
                Me.btnChangeExcd.Hide()

                With Me
                    .t_includeR = workingUList.Item(lb1NewSelect - 3).IncludeR
                    .t_includeQ = workingUList.Item(lb1NewSelect - 3).IncludeQ
                    .t_includeLC = workingUList.Item(lb1NewSelect - 3).IncludeLC
                    .t_allowCallIns = workingUList.Item(lb1NewSelect - 3).AllowCallIns
                    .t_resetDataYN = workingUList.Item(lb1NewSelect - 3).ResetDataYN
                    .t_resetTimeYN = workingUList.Item(lb1NewSelect - 3).ResetTimeYN
                    .t_unitLNum = workingUList.Item(lb1NewSelect - 3).UnitNum
                    .t_unitLocation = workingUList.Item(lb1NewSelect - 3).UnitLocation
                    .t_unitPhoneNum = workingUList.Item(lb1NewSelect - 3).UnitPhoneNum
                    .t_unitOwner = workingUList.Item(lb1NewSelect - 3).UnitOwner
                    .t_unitSerial = workingUList.Item(lb1NewSelect - 3).UnitSerial
                    .t_isEnabled = workingUList.Item(lb1NewSelect - 3).IsEnabled
                    .t_sendCustP = workingUList.Item(lb1NewSelect - 3).SendCustP
                    .t_lockCode = workingUList.Item(lb1NewSelect - 3).LockCode
                    If workingUList.Item(lb1NewSelect - 3).BattVoltage > -1 Then
                        .t_battVoltage = workingUList.Item(lb1NewSelect - 3).BattVoltage.ToString & " V"
                    Else
                        .t_battVoltage = ""
                    End If
                    If workingUList.Item(lb1NewSelect - 3).NumExceedances > -1 Then
                        .t_numExceedances = workingUList.Item(lb1NewSelect - 3).NumExceedances.ToString
                    Else
                        .t_numExceedances = ""
                    End If
                    If workingUList.Item(lb1NewSelect - 3).NumIntervals > -1 Then
                        .t_numIntervals = workingUList.Item(lb1NewSelect - 3).NumIntervals.ToString
                    Else
                        .t_numIntervals = ""
                    End If
                    If workingUList.Item(lb1NewSelect - 3).NumCalibrations > -1 Then
                        .t_numCalibrations = workingUList.Item(lb1NewSelect - 3).NumCalibrations.ToString
                    Else
                        .t_numCalibrations = ""
                    End If
                    If workingUList.Item(lb1NewSelect - 3).NumStartStops > -1 Then
                        .t_numStartStops = workingUList.Item(lb1NewSelect - 3).NumStartStops.ToString
                    Else
                        .t_numStartStops = ""
                    End If
                    If workingUList.Item(lb1NewSelect - 3).CalLevel > -1 Then
                        .t_calLevel = workingUList.Item(lb1NewSelect - 3).CalLevel.ToString & " dB"
                    Else
                        .t_calLevel = ""
                    End If
                    .t_excdThreshold = workingUList.Item(lb1NewSelect - 3).ExcdThreshold
                    .t_lastDL = workingUList.Item(lb1NewSelect - 3).LastDL
                    .t_errorString = workingUList.Item(lb1NewSelect - 3).ErrorString
                    .t_custParams = workingUList.Item(lb1NewSelect - 3).CustParams
                    .t_excdDay = workingUList.Item(lb1NewSelect - 3).ExcdDay
                    .t_excdNight = workingUList.Item(lb1NewSelect - 3).ExcdNight
                End With
        End Select

        UpdateForm()
        lb1OldSelect = lb1NewSelect

    End Sub

    Private Sub UpdateForm()

        cbR.Checked = Me.t_includeR
        cbQ.Checked = Me.t_includeQ
        cbLC.Checked = Me.t_includeLC
        cbCallIn.Checked = Me.t_allowCallIns
        cbResetD.Checked = Me.t_resetDataYN
        cbResetT.Checked = Me.t_resetTimeYN
        cbEn.Checked = Me.t_isEnabled
        cbCP.Checked = Me.t_sendCustP
        tbUN.Text = Me.t_unitLNum
        tbULoc.Text = Me.t_unitLocation
        tbPhone.Text = Me.t_unitPhoneNum
        lblSerial.Text = "Serial: " & Me.t_unitSerial
        lblLock.Text = "Lock Code: " & Me.t_lockCode
        Select Case Me.t_unitOwner
            Case LDUnit.Owner.Aberdeen : Me.cbbOwner.SelectedIndex = 0
            Case LDUnit.Owner.CERL : Me.cbbOwner.SelectedIndex = 1
            Case LDUnit.Owner.Nobody : Me.cbbOwner.SelectedIndex = 2
        End Select
        lblBatt.Text = "Battery Voltage: " & Me.t_battVoltage
        lblE.Text = "Events: " & Me.t_numExceedances
        lblI.Text = "Intervals: " & Me.t_numIntervals
        lblL.Text = "Cals: " & Me.t_numStartStops
        lblC.Text = "Start/Stops: " & Me.t_numCalibrations
        If Date.Compare(Me.t_lastDL, New Date(1901, 1, 1, 0, 0, 0)) > 0 Then
            lblLastDL.Text = "Download Date: " & Me.t_lastDL.ToString("M/d/yyyy h:mm:ss tt")
        Else
            lblLastDL.Text = "Download Date: "
        End If
        If Lb1.SelectedIndex > 2 Then
            If Me.t_excdThreshold > -1 Then
                lblExcd.Text = "Noise Exceedance Threshold: " & Me.t_excdThreshold.ToString & " dB"
            Else
                lblExcd.Text = "Noise Exceedance Threshold: "
            End If
        Else
            lblExcd.Text = "Noise Exceedance Threshold: "
        End If
        lblErr.Text = "Error codes: " & Me.t_errorString
        lblCal.Text = "CaL Level: " & Me.t_calLevel
        If Me.t_excdDay < 0 Then
            tbExcdDay.Text = ""
        Else
            tbExcdDay.Text = Me.t_excdDay.ToString
        End If
        If Me.t_excdNight < 0 Then
            tbExcdNight.Text = ""
        Else
            tbExcdNight.Text = Me.t_excdNight.ToString
        End If
        If lb1NewSelect > 2 AndAlso (workingUList.Item(lb1NewSelect - 3).CalLevel < 80 OrElse _
                workingUList.Item(lb1NewSelect - 3).CalLevel > 110) AndAlso Not _
                workingUList.Item(lb1NewSelect - 3).CalLevel = -1 Then
            Me.lblCal.BackColor = Color.Red
            Me.lblCal.ForeColor = Color.White
            Me.tt.SetToolTip(Me.lblCal, "Warning: Calibration level may be outside normal range. " & vbLf & vbLf & "If the unit also has an unrealistic number of exceedances," & vbLf & "the unit may need servicing.")
        Else
            Me.lblCal.BackColor = System.Windows.Forms.Control.DefaultBackColor
            Me.lblCal.ForeColor = System.Windows.Forms.Control.DefaultForeColor
            Me.tt.SetToolTip(Me.lblCal, "The microphone calibration level for the unit.")
        End If
    End Sub
    Private Function IsAberdeen(ByVal unit As LDUnit) As Boolean
        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Function IsCerl(ByVal unit As LDUnit) As Boolean
        If unit.UnitOwner = LDUnit.Owner.CERL Then
            Return True
        Else
            Return False
        End If
    End Function

    Private Sub OK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles OK.Click
        'save the information for the current selection
        Select Case lb1NewSelect
            Case -1 'do nothing
            Case 0  'write option data to all units in working list
                For Each unit As LDUnit In workingUList
                    'unit.AllowCallIns = t_allowCallIns
                    unit.IncludeR = Me.t_includeR
                    unit.IncludeQ = Me.t_includeQ
                    unit.IncludeLC = Me.t_includeLC
                    unit.ResetDataYN = Me.t_resetDataYN
                    unit.ResetTimeYN = Me.t_resetTimeYN
                    'unit.IsEnabled = Me.t_isEnabled
                    unit.SendCustP = Me.t_sendCustP
                    If Me.t_sendCustP Then
                        unit.CustParams = Me.t_custParams
                        Form1.appset.CustParams = Me.t_custParams
                    End If
                Next
                Form1.appset.WorkdayExcdThresh = Me.t_excdDay
                Form1.appset.OtherExcdThresh = Me.t_excdNight
            Case 1  'write option data to Aberdeen units
                For Each unit As LDUnit In workingUList
                    If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
                        unit.AllowCallIns = t_allowCallIns
                        unit.IncludeR = Me.t_includeR
                        unit.IncludeQ = Me.t_includeQ
                        unit.IncludeLC = Me.t_includeLC
                        unit.ResetDataYN = Me.t_resetDataYN
                        unit.ResetTimeYN = Me.t_resetTimeYN
                        'unit.IsEnabled = Me.t_isEnabled
                        unit.SendCustP = Me.t_sendCustP
                        If Me.t_sendCustP Then
                            unit.CustParams = Me.t_custParams
                            Form1.appset.CustParams = Me.t_custParams
                        End If
                    End If
                Next
            Case 2  'write option data to CERL units
                For Each unit As LDUnit In workingUList
                    If unit.UnitOwner = LDUnit.Owner.CERL Then
                        unit.AllowCallIns = t_allowCallIns
                        unit.IncludeR = Me.t_includeR
                        unit.IncludeQ = Me.t_includeQ
                        unit.IncludeLC = Me.t_includeLC
                        unit.ResetDataYN = Me.t_resetDataYN
                        unit.ResetTimeYN = Me.t_resetTimeYN
                        'unit.IsEnabled = Me.t_isEnabled
                        unit.SendCustP = Me.t_sendCustP
                        If Me.t_sendCustP Then
                            unit.CustParams = Me.t_custParams
                            Form1.appset.CustParams = Me.t_custParams
                        End If
                    End If
                Next
            Case Else   'write all form data to the appropriate unit
                With workingUList.Item(lb1OldSelect - 3)
                    .AllowCallIns = t_allowCallIns
                    .IncludeR = Me.t_includeR
                    .IncludeQ = Me.t_includeQ
                    .IncludeLC = Me.t_includeLC
                    .ResetDataYN = Me.t_resetDataYN
                    .ResetTimeYN = Me.t_resetTimeYN
                    .IsEnabled = Me.t_isEnabled
                    .UnitOwner = Me.t_unitOwner
                    .UnitPhoneNum = Me.t_unitPhoneNum
                    .UnitLocation = Me.t_unitLocation
                    .UnitNum = Me.t_unitLNum
                    .LockCode = Me.t_lockCode
                    .SendCustP = Me.t_sendCustP
                    If Me.t_sendCustP Then
                        .CustParams = Me.t_custParams
                    End If
                    .ExcdDay = Me.t_excdDay
                    .ExcdNight = Me.t_excdNight
                End With
        End Select

        'workingUList is returned in the FormClosing event handler
        Me.DialogResult = Windows.Forms.DialogResult.OK
        Me.Close()
    End Sub

    Private Sub Cancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Cancel.Click
        'ulist is returned in the FormClosing event handler
        Me.DialogResult = Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub

    Private Sub UnitOptions_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        If Me.DialogResult = Windows.Forms.DialogResult.OK Then
            Form1.ReturnAllUnits(workingUList)
        Else
            Form1.ReturnAllUnits(uList)
        End If
    End Sub


    Private Sub cbR_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbR.CheckedChanged
        If cbR.Checked Then
            t_includeR = True
        Else
            t_includeR = False
        End If
    End Sub

    Private Sub cbQ_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbQ.CheckedChanged
        If cbQ.Checked Then
            t_includeQ = True
        Else
            t_includeQ = False
        End If
    End Sub

    Private Sub cbLC_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbLC.CheckedChanged
        If cbLC.Checked Then
            t_includeLC = True
        Else
            t_includeLC = False
        End If
    End Sub

    Private Sub cbCallIn_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbCallIn.CheckedChanged
        If cbCallIn.Checked Then
            Me.t_allowCallIns = True
        Else
            Me.t_allowCallIns = False
        End If
    End Sub

    Private Sub cbResetD_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbResetD.CheckedChanged
```

```vb
        If cbResetD.Checked Then
            Me.t_resetDataYN = True
        Else
            Me.t_resetDataYN = False
        End If
    End Sub

    Private Sub cbResetT_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ↙
    cbResetT.CheckedChanged
        If cbResetT.Checked Then
            Me.t_resetTimeYN = True
        Else
            Me.t_resetTimeYN = False
        End If
    End Sub

    Private Sub cbEn_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbEn.↙
    CheckedChanged
        If cbEn.Checked Then
            Me.t_isEnabled = True
        Else
            Me.t_isEnabled = False
        End If
    End Sub

    Private Sub tbUN_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tbUN.↙
    TextChanged
        Me.t_unitLNum = tbUN.Text
    End Sub


    Private Sub tbULoc_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tbULoc↙
    .TextChanged
        Me.t_unitLocation = tbULoc.Text
    End Sub

    Private Sub tbPhone_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ↙
    tbPhone.TextChanged
        Me.t_unitPhoneNum = tbPhone.Text
    End Sub

    Private Sub cbbOwner_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles cbbOwner.SelectedIndexChanged
        Select Case cbbOwner.SelectedIndex
            Case 0
                Me.t_unitOwner = LDUnit.Owner.Aberdeen
                Me.t_lockCode = "22222222"
            Case 1
                Me.t_unitOwner = LDUnit.Owner.CERL
                Me.t_lockCode = "22222222"
        End Select
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.↙
    Click
        'Add a unit to the working list

        Dim newUnit As New LDUnit(Me.t_unitLNum, Me.t_unitPhoneNum, Me.t_unitOwner)

        Me.cbEn.Checked = True 'enables the new unit - usually desired

        With newUnit
            .IncludeR = Me.t_includeR
            .IncludeQ = Me.t_includeQ
            .IncludeLC = Me.t_includeLC
            .AllowCallIns = Me.t_allowCallIns
            .ResetDataYN = Me.t_resetDataYN
            .ResetTimeYN = Me.t_resetTimeYN
            .IsEnabled = True 'usually desired when adding a unit
            .UnitLocation = Me.t_unitLocation
        End With

        workingUList.Add(newUnit)
        Lb1.Items.Add(newUnit.UnitNum)

    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.↙
    Click
        'Remove a unit from the working list.
        'MessageBox.Show(Me.lb1NewSelect.ToString)
        Dim dres As DialogResult
        Dim sel As Integer = lb1NewSelect

        If lb1NewSelect > 2 Then
            dres = MessageBox.Show("Remove unit " & Me.t_unitLNum & "?", "Delete unit", _
            MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)
            'MessageBox.Show(Me.lb1NewSelect.ToString)
            If dres = Windows.Forms.DialogResult.Yes Then
                Lb1.Items.RemoveAt(sel)
                workingUList.RemoveAt(sel - 3)
            End If
        End If
    End Sub

    Private Sub cbCP_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cbCP.
    CheckedChanged
        If cbCP.Checked Then
            Me.t_sendCustP = True
        Else
            Me.t_sendCustP = False
        End If
    End Sub

    Private Sub btnCust_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCust.
    Click
        Dim cp As New CustomParamsDialog
        cp.TextBox1.Text = Me.t_custParams
        Dim dres As DialogResult = cp.ShowDialog()
        If dres = Windows.Forms.DialogResult.OK Then
            Me.t_custParams = cp.TextBox1.Text
            cbCP.Checked = True
        End If
    End Sub

    Private Sub tbExcdDay_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    tbExcdDay.TextChanged
        'makes sure the text entered is a number between 0 and 1000
        Dim res As Integer
        Dim tryp As Boolean = Integer.TryParse(tbExcdDay.Text, res)

        If tryp = True AndAlso res > 29 AndAlso res < 1000 Then
            Me.t_excdDay = res
            If tbExcdDay.Text.Length <= 1 Then
                Me.cbResetD.Checked = True
            End If
        End If
    End Sub


    Private Sub tbExcdNight_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    tbExcdNight.TextChanged
        'makes sure the text entered is a number between 0 and 1000
        Dim res As Integer
        Dim tryp As Boolean = Integer.TryParse(tbExcdNight.Text, res)

        If tryp = True AndAlso res > 29 AndAlso res < 1000 Then
            Me.t_excdNight = res
            If tbExcdNight.Text.Length <= 1 Then
                Me.cbResetD.Checked = True
            End If
        End If
    End Sub

    Private Sub btnChangeExcd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    btnChangeExcd.Click
        'writes day and night exceedance values to all of the units

        For Each unit As LDUnit In workingUList
            unit.ExcdDay = Me.t_excdDay
            unit.ExcdNight = Me.t_excdNight
        Next

    End Sub
End Class
```

Appendix F: Unit Viewer



Unit Info

Number: Owner:

Location: Phone:

Noise Exceedance Threshold: Battery Voltage:

Cal Level: Serial:

Lock Code:

Statistics for Last Download

Events: Intervals: Cals:

Download Date: Starts/Stops:

Error Codes:

Options

Include Read Parameters

Include Q Parameters

Include Log and Cal Parameters

Allow Unit to Dial In

Reset Data

Reset Time

```
Imports System.IO

Public Class UnitView
    Private workingUList As New List(Of LDUnit)
    Private uList As New List(Of LDUnit)
    Private lb1OldSelect As Integer
    Private lb1NewSelect As Integer
    Private lb1NewSelect2 As Integer
    Private ucomp As New uCompare()
    Private comp As Collections.Generic.IComparer(Of LDUnit) = ucomp
    Private currLNum As String

    Protected t_unitSerial As String = ""
    Protected t_unitLNum As String = ""
    Protected t_unitLocation As String = ""
    Protected t_unitPhoneNum As String = ""
    Protected t_unitOwner As LDUnit.Owner
    Protected t_lockCode As String = ""
    Protected t_lastDL As DateTime

    Protected t_includeR As Boolean
    Protected t_includeLC As Boolean
    Protected t_includeQ As Boolean
    Protected t_allowCallIns As Boolean
    Protected t_resetDataYN As Boolean
    Protected t_resetTimeYN As Boolean
    Protected t_isEnabled As Boolean
    Protected t_sendCustP As Boolean

    Protected t_numExceedances As String = ""
    Protected t_numIntervals As String = ""
    Protected t_numStartStops As String = ""
    Protected t_numCalibrations As String = ""
    Protected t_battVoltage As String = ""
    Protected t_errorString As String = ""
    Protected t_excdThreshold As Integer = 0
    Public t_custParams As String = ""
    Protected t_calLevel As String = ""

    Private Sub UnitView_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'load a reference list from disk
        Try
            Using fs As New FileStream(Form1.appset.InstallPath & "units.dat", FileMode.Open)

                Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter
                uList = DirectCast(bf.Deserialize(fs), List(Of LDUnit))
                'Deserialize returns an object, which is then cast to a List(Of LDUnit)

            End Using
        Catch ex As FileNotFoundException   'file does not exist - shouldn't load this form
            MessageBox.Show("Can't open Unit Viewer - File " & Form1.appset.InstallPath & "units.dat not ↙
    found", _
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            Me.btnClose.PerformClick()
        End Try

        Populate()

        lb1OldSelect = -1
        lb1NewSelect = -1
        lb1NewSelect2 = -1

        Me.DialogResult = Windows.Forms.DialogResult.None
        Me.tsslInProgress.Text = ""
        Me.tsslUnitsLeft.Text = ""

        tt.SetToolTip(Me.lblEn, "Indicates whether the program will dial out to the unit.")
        tt.SetToolTip(Me.lblC, "The number of self-calibrations performed by the unit since the last data ↙
    reset.")
        tt.SetToolTip(Me.lblErr, "The most recent error is at the top of this list.")

        'Me.lblC.BorderStyle = BorderStyle.FixedSingle
        'Me.lblC.BackColor = Color.Red
        'Me.lblC.ForeColor = Color.White



    End Sub

    Private Sub UnitView_Shown(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Shown
        Me.RefreshToolStripMenuItem.PerformClick()

    End Sub

    Private Sub Populate()
        'loads the current LDUnitList in RAM and populates the list box
        Dim idx As Integer = -1
        Dim inProgList As New System.Text.StringBuilder()

        'copy contents of current LDUnitList
        workingUList = Form1.ReadLDUnitList()
        workingUList.Sort(comp)

        'handle the case when units are missing from the current LDUnitList
        'because a download is in progress
        '(uList is assumed to be the master list, containing one of every unit,
        'while workingUList has all of the current data)
        For Each u As LDUnit In uList
            Me.currLNum = u.UnitNum
            If Not Me.workingUList.Exists(AddressOf SpecUnit) Then 'unit is checked out
                Dim u2 As LDUnit = u
                u2.LastDL = New Date(6, 6, 6)
                workingUList.Add(u2)
                If inProgList.Length < 1 Then
                    inProgList.Append(u2.UnitNum)
                Else
                    inProgList.Append(", " & u2.UnitNum)
                End If
                'else do nothing - the unit is there and more current than the unit in the list on disk
            End If
        Next

        'populate the checked list box
        workingUList.Sort(comp)
        Lb1.Items.Clear()
        Lb1.BeginUpdate()
        For Each unit As LDUnit In workingUList
            Lb1.Items.Add(unit.UnitNum)
        Next
        Lb1.EndUpdate()

        If inProgList.Length > 3 Then
            Me.tsslInProgress.Text = "Now downloading units " & inProgList.ToString
        ElseIf inProgList.Length > 0 Then
            Me.tsslInProgress.Text = "Now downloading unit " & inProgList.ToString
        Else
            Me.tsslInProgress.Text = ""
        End If

    End Sub

    Private Function SpecUnit(ByVal unit As LDUnit) As Boolean
        If String.Compare(unit.UnitNum, Me.currLNum) = 0 Then
            Return True
        Else
            Return False
        End If
    End Function

    Private Sub Lb1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ↙
    Lb1.SelectedIndexChanged
        lb1NewSelect = Lb1.SelectedIndex

        Select Case lb1NewSelect
            Case -1 'do nothing
            Case Else   'load data from the selected unit to the class vars and the UI
                gbUnitInfo.Enabled = True
                gbStat.Enabled = True
                With Me
                    .t_includeR = workingUList.Item(lb1NewSelect).IncludeR
                    .t_includeQ = workingUList.Item(lb1NewSelect).IncludeQ
                    .t_includeLC = workingUList.Item(lb1NewSelect).IncludeLC
                    .t_allowCallIns = workingUList.Item(lb1NewSelect).AllowCallIns
                    .t_resetDataYN = workingUList.Item(lb1NewSelect).ResetDataYN
                    .t_resetTimeYN = workingUList.Item(lb1NewSelect).ResetTimeYN
                    .t_unitLNum = workingUList.Item(lb1NewSelect).UnitNum
                    .t_unitLocation = workingUList.Item(lb1NewSelect).UnitLocation
                    .t_unitPhoneNum = workingUList.Item(lb1NewSelect).UnitPhoneNum
                    .t_unitOwner = workingUList.Item(lb1NewSelect).UnitOwner
                    .t_unitSerial = workingUList.Item(lb1NewSelect).UnitSerial
                    .t_isEnabled = workingUList.Item(lb1NewSelect).IsEnabled
                    .t_sendCustP = workingUList.Item(lb1NewSelect).SendCustP
                    .t_lockCode = workingUList.Item(lb1NewSelect).LockCode
                    If workingUList.Item(lb1NewSelect).BattVoltage > -1 Then
                        .t_battVoltage = workingUList.Item(lb1NewSelect).BattVoltage.ToString & " V"
                    Else
                        .t_battVoltage = ""
                    End If
                    If workingUList.Item(lb1NewSelect).NumExceedances > -1 Then
                        .t_numExceedances = workingUList.Item(lb1NewSelect).NumExceedances.ToString
                    Else
                        .t_numExceedances = ""
                    End If
                    If workingUList.Item(lb1NewSelect).NumIntervals > -1 Then
                        .t_numIntervals = workingUList.Item(lb1NewSelect).NumIntervals.ToString
                    Else
                        .t_numIntervals = ""
                    End If
                    If workingUList.Item(lb1NewSelect).NumCalibrations > -1 Then
                        .t_numCalibrations = workingUList.Item(lb1NewSelect).NumCalibrations.ToString
                    Else
                        .t_numCalibrations = ""
                    End If
                    If workingUList.Item(lb1NewSelect).NumStartStops > -1 Then
```

```
                        .t_numStartStops = workingUList.Item(lb1NewSelect).NumStartStops.ToString
                    Else
                        .t_numStartStops = ""
                    End If
                    If workingUList.Item(lb1NewSelect).CalLevel > -1 Then
                        .t_calLevel = workingUList.Item(lb1NewSelect).CalLevel.ToString & " dB"
                    Else
                        .t_calLevel = ""
                    End If
                    .t_excdThreshold = workingUList.Item(lb1NewSelect).ExcdThreshold
                    .t_lastDL = workingUList.Item(lb1NewSelect).LastDL
                    .t_errorString = workingUList.Item(lb1NewSelect).ErrorString
                    .t_custParams = workingUList.Item(lb1NewSelect).CustParams
                End With
        End Select

        UpdateForm()
        lb1OldSelect = lb1NewSelect

    End Sub

    Private Sub UpdateForm()

        If Me.t_includeR Then
            lblR.Text = "Include Read Parameters: Yes"
        Else
            lblR.Text = "Include Read Parameters: No"
        End If
        If Me.t_includeQ Then
            lblQ.Text = "Include Q Parameters: Yes"
        Else
            lblQ.Text = "Include Q Parameters: No"
        End If
        If Me.t_includeLC Then
            lblLC.Text = "Include Log and Cal Parameters: Yes"
        Else
            lblLC.Text = "Include Log and Cal Parameters: No"
        End If
        If Me.t_allowCallIns Then
            lblCallIn.Text = "Allow Unit to Dial In: Yes"
        Else
            lblCallIn.Text = "Allow Unit to Dial In: No"
        End If
        If Me.t_resetDataYN Then
            lblResetD.Text = "Reset Data: Yes"
        Else
            lblResetD.Text = "Reset Data: No"
        End If
        If Me.t_resetTimeYN Then
            lblResetT.Text = "Reset Time: Yes"
        Else
            lblResetT.Text = "Reset Time: No"
        End If
        If Me.t_isEnabled Then
            lblEn.Text = "Enabled: Yes"
        Else
            lblEn.Text = "Enabled: No"
        End If
        lblUN.Text = "ID Number: " & Me.t_unitLNum
        lblULoc.Text = "Location: " & Me.t_unitLocation
        lblPhone.Text = "Phone: " & Me.t_unitPhoneNum
        lblSerial.Text = "Serial: " & Me.t_unitSerial
        lblLock.Text = "Lock Code: " & Me.t_lockCode
        Select Case Me.t_unitOwner
            Case LDUnit.Owner.Aberdeen : Me.lblOwner.Text = "Owner: Aberdeen"
            Case LDUnit.Owner.CERL : Me.lblOwner.Text = "Owner: CERL"
            Case LDUnit.Owner.Nobody : Me.lblOwner.Text = "Owner: ???"
        End Select
        lblBatt.Text = "Battery Voltage: " & Me.t_battVoltage
        lblE.Text = "Events: " & Me.t_numExceedances
        lblI.Text = "Intervals: " & Me.t_numIntervals
        lblL.Text = "Start/Stops: " & Me.t_numStartStops
        lblC.Text = "Cals: " & Me.t_numCalibrations
        If Date.Compare(Me.t_lastDL, New Date(1901, 1, 1, 0, 0, 0)) > 0 Then
            lblLastDL.Text = "Download Date: " & Me.t_lastDL.ToString("M/d/yyyy h:mm:ss tt")
            lblErr2.Text = "Error Codes" & vbLf & "as of " & vbLf & Me.t_lastDL.ToString("M/d/yy") & " :"
        ElseIf Me.t_lastDL = New Date(6, 6, 6) Then
            lblLastDL.Text = "Download Date: In Progress"
            lblErr2.Text = "Error Codes:"
        Else
            lblLastDL.Text = "Download Date: "
            lblErr2.Text = "Error Codes:"
        End If
        If Me.t_excdThreshold > -1 Then
            lblExcd.Text = "Noise Exceedance Threshold: " & Me.t_excdThreshold.ToString & " dB"
        Else
            lblExcd.Text = "Noise Exceedance Threshold: "
        End If
        lblCal.Text = "Cal Level: " & Me.t_calLevel
        If (workingUList.Item(lb1NewSelect).CalLevel < 80 OrElse _
        workingUList.Item(lb1NewSelect).CalLevel > 110) AndAlso Not _
        workingUList.Item(lb1NewSelect).CalLevel = -1 Then
```

```vb
            Me.lblCal.BackColor = Color.Red
            Me.lblCal.ForeColor = Color.White
            Me.tt.SetToolTip(Me.lblCal, "Warning: Calibration level may be outside normal range. " & vbLf & vbLf & "If the unit also has an unrealistic number of exceedances," & vbLf & "the unit may need servicing.")
        Else
            Me.lblCal.BackColor = System.Windows.Forms.Control.DefaultBackColor
            Me.lblCal.ForeColor = System.Windows.Forms.Control.DefaultForeColor
            Me.tt.SetToolTip(Me.lblCal, "The microphone calibration level for the unit.")
        End If

        'create verbose error codes
        Dim trimChars3() As Char = {","c}
        Dim esTemp() As String = t_errorString.Split(trimChars3, StringSplitOptions.RemoveEmptyEntries)
        Dim errorInts(esTemp.Length - 1) As Integer
        Dim v As String = ""

        Me.t_errorString = ""

        For i As Integer = 0 To esTemp.Length - 1
            Dim tryp As Boolean = Integer.TryParse(esTemp(i), errorInts(i))
            If tryp Then 'map a line of text to the numeric code
                Select Case errorInts(i)
                    Case 1 : v = "Out of Memory"
                    Case 2 : v = "Battery Low"
                    Case 3 : v = "Power Failure"
                    Case 4 : v = "Division by Zero"
                    Case 5 : v = "Operand-1 Range (safe to ignore)"
                    Case 6 : v = "Operand-2 Range (safe to ignore)"
                    Case 7 : v = "DPC Format"
                    Case 8 : v = "Key Has No Effect"
                    Case 9 : v = "Stop Required (safe to ignore)"
                    Case 10 : v = "Key Has No Effect In ""View"""
                    Case 11 : v = "Parameter Entered Wrong (safe to ignore)"
                    Case 12 : v = "RESET-ALL Required"
                    Case 13 : v = "Use Arrows, (ON) to Exit"
                    Case 14 : v = "Use NEXT/PREV or ENTER"
                    Case 15 : v = "Invalid Numeric Entry"
                    Case 16 : v = "Open # (hey, I just report it)"
                    Case 17 : v = "Already Open"
                    Case 18 : v = "No History Yet"
                    Case 19 : v = "At End of History"
                    Case 20 : v = "At Start of History"
                    Case 21 : v = "History Format Error"
                    Case 22 : v = "Unknown I/O Command"
                    Case 23 : v = "I/O Operand Invalid"
                    Case 24 : v = "Unable to Calibrate"
                    Case 25 : v = "EEPROM Write Error"
                    Case 26 : v = "Memory was Lost, Data Reset (check all batteries)"
                    Case 27 : v = "RECALL- Not Found"
                    Case 28 : v = "Function Not Implemented"
                    Case 29 : v = "System Locked"
                    Case 30 : v = "A:D Stack Full"
                    Case 31 : v = "A:D Overrun"
                    Case 32 : v = "RS-232 Framing Error"
                    Case 33 : v = "RS-232 Line Noisy"
                    Case 34 : v = "RS-232 Overrun (safe to ignore)"
                    Case 35 : v = "No Error (truly, a contradiction)"
                    Case 36 : v = "Power Turned Off"
                    Case 37 : v = "Time/Date Not Set"
                    Case 38 : v = "Whatever Old-Ass Printer You Connected to this thing is Busy"
                    Case 39 : v = "Lithium (backup) Battery Low"
                    Case 40 : v = "Timer ON Pending"
                    Case 41 : v = "External Power Failure"
                    Case 42 : v = "Calibration Changed"
                    Case 43 : v = "I/O Buffer Overflow"
                    Case 44 : v = "Crack Overdose - Treatment Needed"
                    Case 101 : v = "DEVICE ERROR: Count Overflow"
                    Case 102 : v = "DEVICE ERROR: Exponential Overflow"
                    Case 103 : v = "DEVICE ERROR: RTX Task Select"
                    Case 104 : v = "DEVICE ERROR: Illegal Exchange Rate"
                    Case 105 : v = "DEVICE ERROR: Unknown Interrupt"
                    Case 106 : v = "DEVICE ERROR: Watchdog Reset"
                    Case 107 : v = "DEVICE ERROR: RAM Bank Selection Error"
                    Case 108 : v = "DEVICE ERROR: Opcode Error"
                End Select
                Me.t_errorString &= errorInts(i).ToString & " - " & v & vbLf
            End If
        Next
        lblErr.Text = Me.t_errorString

    End Sub

    Private Function IsAberdeen(ByVal unit As LDUnit) As Boolean
        If unit.UnitOwner = LDUnit.Owner.Aberdeen Then
            Return True
        Else
            Return False
        End If
    End Function
    Private Function IsCerl(ByVal unit As LDUnit) As Boolean
        If unit.UnitOwner = LDUnit.Owner.CERL Then
```

```
            Return True
        Else
            Return False
        End If
    End Function


#Region "Menu Strip Items"
    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles ExitToolStripMenuItem.Click
        Me.btnClose.PerformClick()
    End Sub

    Private Sub SaveToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles SaveToolStripMenuItem.Click

        Me.SaveFileDialog2.InitialDirectory = Form1.appset.InstallPath & "\Logs"
        Me.SaveFileDialog2.FileName = "Unit Info " & Date.Now.ToString("ddMMMyyyy HHmm")
        Me.SaveFileDialog2.ShowDialog(Me)

    End Sub

    Private Sub SaveFileDialog2_FileOk(ByVal sender As System.Object, ByVal e As System.ComponentModel. ↙
    CancelEventArgs) Handles SaveFileDialog2.FileOk
        'Save the unit data to a CSV file (and eventually to a database table)
        'first line should be the column headers
        'subsequent lines should be a row for each unit

        Dim unitLog As New System.Text.StringBuilder()
        Dim CSB As String = "    <Cell><Data ss:Type=""String"">"
        Dim CNB As String = "    <Cell><Data ss:Type=""Number"">"
        Dim CE As String = "</Data></Cell>"
        Dim aci As String = ""
        Dim rd As String = ""
        Dim rt As String = ""
        Dim r As String = ""
        Dim q As String = ""
        Dim lc As String = ""
        Dim lastDLstr As String = ""

        Using sr1 As New StreamReader(Form1.appset.InstallPath & "sampless_begin.txt")
            unitLog.Append(sr1.ReadToEnd())
        End Using

        unitLog.Append((workingUList.Count + 1).ToString)

        Using sr2 As New StreamReader(Form1.appset.InstallPath & "sampless_middle.txt")
            unitLog.Append(sr2.ReadToEnd())
        End Using

        workingUList.Sort(comp)

        For Each unit As LDUnit In workingUList

            If unit.LastDL < New Date(1901, 1, 1) Then
                lastDLstr = ""
            Else
                lastDLstr = unit.LastDL.ToString("M/d/yyyy HH:mm:ss")
            End If
            If unit.AllowCallIns Then
                aci = "Yes"
            Else
                aci = "No"
            End If
            If unit.ResetDataYN Then
                rd = "Yes"
            Else
                rd = "No"
            End If
            If unit.ResetTimeYN Then
                rt = "Yes"
            Else
                rt = "No"
            End If
            If unit.IncludeR Then
                r = "Yes"
            Else
                r = "No"
            End If
            If unit.IncludeQ Then
                q = "Yes"
            Else
                q = "No"
            End If
            If unit.IncludeLC Then
                lc = "Yes"
            Else
                lc = "No"
            End If

            unitLog.AppendLine("   <Row>")
            unitLog.AppendLine(CSB & unit.UnitNum & CE)
            unitLog.AppendLine(CSB & lastDLstr & CE)
            unitLog.AppendLine(CSB & aci & CE)
            unitLog.AppendLine(CSB & rd & CE)
            unitLog.AppendLine(CSB & rt & CE)

            If unit.NumExceedances > -1 Then
                unitLog.AppendLine(CNB & unit.NumExceedances.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If
            If unit.NumIntervals > -1 Then
                unitLog.AppendLine(CNB & unit.NumIntervals.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If
            If unit.NumStartStops > -1 Then
                unitLog.AppendLine(CNB & unit.NumStartStops.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If
            If unit.NumCalibrations > -1 Then
                unitLog.AppendLine(CNB & unit.NumCalibrations.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If

            If unit.ExcdThreshold > -1 Then
                unitLog.AppendLine(CNB & unit.ExcdThreshold.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If
            If unit.CalLevel > -1 Then
                unitLog.AppendLine(CNB & unit.CalLevel.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If
            If unit.BattVoltage > -1 Then
                unitLog.AppendLine(CNB & unit.BattVoltage.ToString & CE)
            Else
                unitLog.AppendLine(CSB & CE)
            End If

            unitLog.AppendLine(CSB & unit.ErrorString & CE)
            unitLog.AppendLine(CSB & r & CE)
            unitLog.AppendLine(CSB & q & CE)
            unitLog.AppendLine(CSB & lc & CE)
            unitLog.AppendLine("   </Row>")

        Next

        Using sr3 As New StreamReader(Form1.appset.InstallPath & "sampless_end.txt")
            unitLog.Append(sr3.ReadToEnd())
        End Using

        Using sw1 As New StreamWriter(Me.SaveFileDialog2.FileName)
            sw1.Write(unitLog.ToString())
        End Using

    End Sub
    Private Sub RefreshToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↙
    Handles RefreshToolStripMenuItem.Click
        'stuff to do to refresh the display
        'display is refreshed whenever a unit is checked in or out of the LDUnitList

        lb1NewSelect2 = Lb1.SelectedIndex
        Populate()
        If lb1NewSelect2 > -1 Then
            Lb1.SelectedIndex = lb1NewSelect2
        End If

    End Sub

#End Region


    Private Sub Cancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnClose. ↙
    Click
        Me.DialogResult = Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub

    Private Sub UnitView_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms. ↙
    FormClosingEventArgs) Handles Me.FormClosing
        Form1.IsUVOpen = False 'note that this writes to a _property_, not a field

        If e.CloseReason = CloseReason.UserClosing OrElse e.CloseReason = CloseReason.None Then
            'cancel the form closing and disposal - instead, just hide the form
            e.Cancel = True
            Me.Hide()
        End If
        'don't return (add) any units here to the LDUnitList
    End Sub

End Class
```

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.			
1. REPORT DATE (DD-MM-YYYY) 05-2007	**2. REPORT TYPE** Final		**3. DATES COVERED (From - To)**
4. TITLE AND SUBTITLE ERDC-CERL LD-870 Download Program: Programming Manual			**5a. CONTRACT NUMBER** MIPR6FXXR3A563
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Ben Niemoeller and Edward T. Nykaza			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER 0B72D9
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Engineer Research and Development Center (ERDC) Construction Engineering Research Laboratory (CERL) PO Box 9005 Champaign, IL 61826-9005			**8. PERFORMING ORGANIZATION REPORT NUMBER** ERDC/CERL SR-07-7
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Commander, U.S. Army Aberdeen Test Center 400 Colleran Road Aberdeen Proving Ground, MD 21005-5059			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			
13. SUPPLEMENTARY NOTES			

14. ABSTRACT

The U.S. Army Engineer Research and Development Center Construction Engineering Research Laboratory has developed software that interfaces with an array of Larson-Davis Model 870 Environmental Noise Monitors for Aberdeen Test Center. This document explains logic and procedures used while programming the software that are of interest to a programmer looking to modify or expand the functionality of the program. The following topic areas are covered: terminology, time synchronization, and scheduling events. This document will be of interest to those who wish to modify the L-D Download software program. The code, which was written with Microsoft Visual Studio 2005, is included in the appendices.

15. SUBJECT TERMS

Programming manual, noise monitor, military training, Aberdeen Test Center, MD, environmental management, data management

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	SAR		**19b. TELEPHONE NUMBER (include area code)**